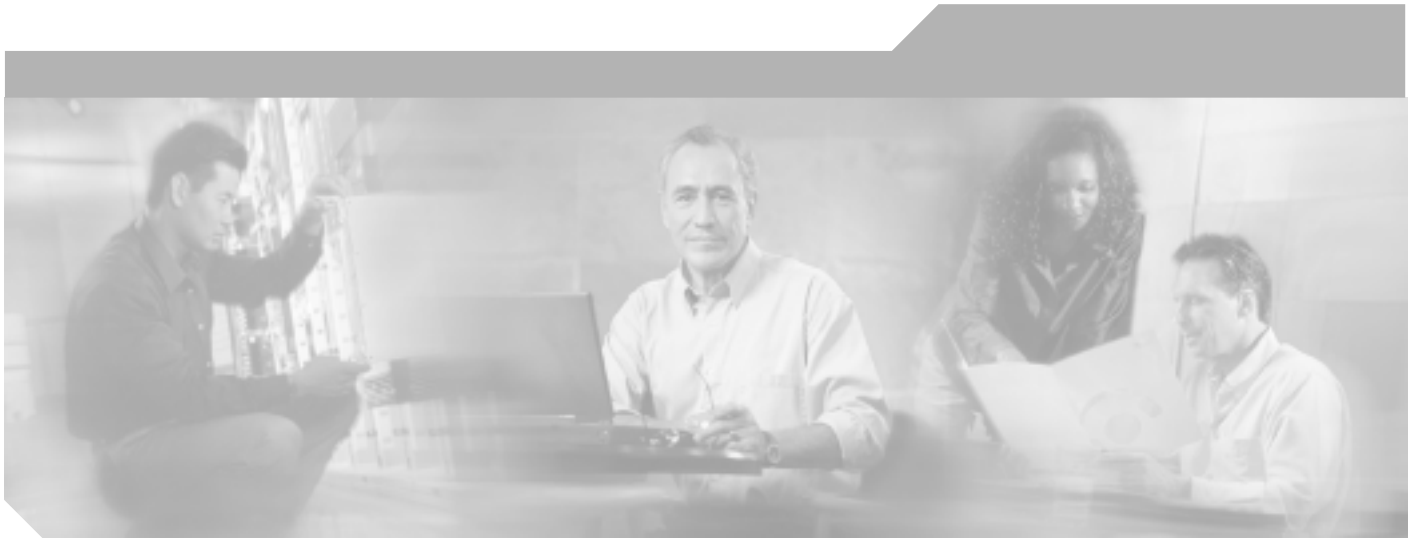


Service Control Application Suite for Broadband API プログラマ ガイド

Ver. 2.5.5

Text Part Number: OL-7207-01-J

このマニュアルに記載されている仕様および製品に関する情報は、予告なしに変更されることがあります。このマニュアルに記載されている表現、情報、および推奨事項は、すべて正確であると考えていますが、明示的であれ黙示的であれ、一切の保証の責任を負わないものとします。このマニュアルに記載されている製品の使用は、すべてユーザ側の責任になります。

対象製品のソフトウェア ライセンスおよび限定保証は、製品に添付された『Information Packet』に記載されています。添付されていない場合には、代理店にご連絡ください。

FCC クラス A 適合装置に関する記述：この装置はテスト済みであり、FCC ルール Part 15 に規定された仕様のクラス A デジタル装置の制限に適合していることが確認済みです。これらの制限は、商業環境で装置を使用したときに、干渉を防止する適切な保護を規定しています。この装置は、無線周波エネルギーを生成、使用、または放射する可能性があり、この装置のマニュアルに記載された指示に従って設置および使用しなかった場合、ラジオおよびテレビの受信障害が起こることがあります。住宅地でこの装置を使用すると、干渉を引き起こす可能性があります。その場合には、ユーザ側の負担で干渉防止措置を講じる必要があります。

FCC クラス B 適合装置に関する記述：このマニュアルに記載された装置は、無線周波エネルギーを生成および放射する可能性があります。シスコシステムズの指示する設置手順に従わずに装置を設置した場合、ラジオおよびテレビの受信障害が起こることがあります。この装置はテスト済みであり、FCC ルール Part 15 に規定された仕様のクラス B デジタル装置の制限に適合していることが確認済みです。これらの仕様は、住宅地で使用したときに、このような干渉を防止する適切な保護を規定したものです。ただし、特定の設置条件において干渉が起きないことを保証するものではありません。

シスコシステムズの書面による許可なしに装置を改造すると、装置がクラス A またはクラス B のデジタル装置に対する FCC 要件に適合しなくなることがあります。その場合、装置を使用するユーザの権利が FCC 規制により制限されることがあり、ラジオまたはテレビの通信に対するいかなる干渉もユーザ側の負担で矯正するように求められることがあります。

装置の電源を切ることによって、この装置が干渉の原因であるかどうかを判断できます。干渉がなくなれば、シスコシステムズの装置またはその周辺機器が干渉の原因になっていると考えられます。装置がラジオまたはテレビ受信に干渉する場合には、次の方法で干渉が起きないようにしてください。

・干渉がなくなるまで、テレビまたはラジオのアンテナの向きを変えます。

・テレビまたはラジオの左右どちらかの側に装置を移動させます。

・テレビまたはラジオから離れたところに装置を移動させます。

・テレビまたはラジオとは別の回路にあるコンセントに装置を接続します（装置とテレビまたはラジオがそれぞれ別個のブレーカーまたはヒューズで制御されるようにします）。

米国シスコシステムズ社では、この製品の変更または改造を認めていません。変更または改造した場合には、FCC 認定が無効になり、さらに製品を操作する権限を失うことになります。

シスコシステムズが採用している TCP ヘッダー圧縮機能は、UNIX オペレーティング システムの UCB（University of California, Berkeley）パブリック ドメイン バージョンの一部として、UCB が開発したプログラムを最適化したものです。

ここに記載されている他のいかなる保証にもよらず、各社のすべてのマニュアルおよびソフトウェアは、障害も含めて「現状のまま」として提供されます。シスコシステムズおよびこれら各社は、商品性や特定の目的への適合性、権利を侵害しないことに関する、または取り扱い、使用、または取引によって発生する、明示されたまたは黙示された一切の保証の責任を負わないものとします。

いかなる場合においても、シスコシステムズおよびその代理店は、このマニュアルの使用またはこのマニュアルを使用できないことによって起こる制約、利益の損失、データの損傷など間接的で偶発的に起こる特殊な損害のあらゆる可能性がシスコシステムズまたは代理店に知らされていても、それらに対する責任を一切負いかねます。

Service Control Application Suite for Broadband API プログラマ ガイド

CONTENTS

FIGURES

はじめに

このマニュアルでは、SCAS BB アプリケーションの API のセットを使用してプログラムを作成し、使用する方法を説明します。

対象読者

このマニュアルは、経験豊富な Java プログラマ向けに書かれています。読者はネットワーク プロトコルおよびトポロジーに関する実務的な知識があることを前提としています。

SCAS BB の開発者は、SCAS Service Configuration API クラス、SCAS Reporter CLI（コマンドライン インターフェイス）のほか、メンテナンス スクリプト、データベース スクリプトなどの SCAS BB ツールを使用して付加価値アプリケーションを作成し、プロバイダー ネットワーク上の既存のハードウェアおよびソフトウェアに統合し、付加価値ソフトウェア ソリューションおよびサービスを提供することができます。

目的

この『*Service Control Application Suite for Broadband API プログラマ ガイド*』では、サービス プロバイダー（SP）によるネットワーク加入者向け付加価値サービスの監視およびプロビジョニングを可能にする、サービス コントロール テクノロジーおよびソリューションを使用したプログラムの作成と使用方法を説明します。

Service Control Engine（SCE）プラットフォームをベースとするサービス コントロール ソリューションは、詳細なネットワーク解析、トラフィック シェーピングと優先順位付け、およびトランザクション レベルの制御を、すべてワイヤ速度で、完全にプログラマブルかつ拡張性のあるフレームワークで実行できる、柔軟性に富んだ使いやすい環境をサービス プロバイダーに提供します。SP はこれらの機能を利用して、ネットワークの使用状況を把握し、使用状況に基づく高度な課金方式を開発し、ネットワークの不正使用を防止し、階層型のアクセス サービスを提供することができます。

SCAS BB API テクノロジーを使用すると、基盤となる複雑なレイヤ構造を隠し、ネットワーク ユーザに差別化サービスを提供する、ネットワーク型 Java アプリケーションを簡単に開発して展開することができます。新規または既存の OSS およびバックオフィス システムとの統合により、サービス コントロール ソリューションの機能を拡張できます。API を使用して、基盤となる複雑なレイヤ構造を隠し、ネットワーク ユーザに差別化サービスを提供する、豊富な機能の集合を開発して展開することもできます。

このマニュアルの各章を順に参照することにより、SCAS BB API によるアプリケーションの開発方法を学習できます。SCAS BB API は Java で実装されているので、プラットフォームに依存しない、簡素で相互運用可能なプログラミング環境を提供します。

SCAS BB API ソフトウェア製品を使用するには、読者（システム管理者）はこれまでにある種の管理作業（プロバイダー ネットワークへの Service Control システム ハードウェアおよびソフトウェアのインストールなど）を行った経験が必要です。SP ネットワーク管理者は、IP アドレスの変更、ネットワークへのコンピュータおよび Service Control ハードウェア デバイスの追加などの業務を担当します。

（注） SCAS BB Service Configuration API を使用してアプリケーションを開発するには、**JDK** バージョン 1.3 をインストールして使用する必要があります。

マニュアルの構成

このマニュアルは、次の章で構成されています。

第 1 章「*概要*」— サービス コントロール ソリューションの概要、およびサービス コントロールの概念を示します。

第 2 章「*サービス コントロール ソリューション*」— サービス コントロール テクノロジーおよびソリューションのさまざまな側面（システム コンポーネント、サブスクライバおよびサブスクライバ モード、サービス コンフィギュレーションなど）について説明します。

第 3 章「*開発者のためのシステム アーキテクチャ*」— Service Control Application Suite for Broadband のアーキテクチャについて、プログラマ向けに概要を示します。

第 4 章「*サービス コンフィギュレーション エンティティ*」— Service Control Application Suite for Broadband アプリケーションの開発について、定義および原則を示します。

第 5 章「*Service Configuration API*」— SCAS BB Service Configuration API を使用して、SCAS BB サービスの設定を自動的に行う方法を説明します。

第 6 章「*サブスクライバ統合*」— Service Control Application Suite for Broadband アプリケーションにおけるサブスクライバの統合について説明します。

第 7 章「*Quota Provisioning API*」— Java および C に対応する外部 Quota Provisioning（QP）API について説明します。

第 8 章「*Reporter CLI*」— Reporter CLI を使用してレポートを生成する方法を説明します。

用語集

索引

表記法

このマニュアルでは、次の表記法を使用しています。

書体または記号	意味
イタリック体	参照先、新出用語、フィールド名、およびプレースホルダ
太字	メニュー名、オプション名、およびコマンド ボタン名
`Courier`	Telnet セッションでコンピュータ画面に表示されるシステム出力
太字の `Courier`	例示において、ユーザが入力する CLI（コマンドライン インターフェイス）コード
イタリック体の `Courier`	CLI コードの必須パラメータ
[*角カッコで囲まれたイタリック体*]	CLI コードの省略可能なパラメータ
	「*注釈*」です。 重要な情報が紹介されています。
	「*警告*」です。 人身事故または機器損傷の危険性があることを意味します。

関連資料

この『*SCAS BB API プログラマ ガイド*』は、SCE プラットフォームのユーザ ガイド（『*SCE1000 User Guide*』、『*SCE3000 User Guide*』）およびその他の Management Suite ユーザ ガイド（『*Collection Manager User Guide*』、『*SCAS BB User Guide*』、『*smartSUB User Guide*』、および『*SM API Guide*』）と併せて使用してください。

テクニカル サポート

Technical Assistance Center（TAC）では、シスコシステムズとサービス契約を結んでいるお客様、パートナー、リセラー、販売店を対象として、評価の高い 24 時間体制のテクニカル サポート サービスをオンラインおよび電話で提供しています。Cisco.com では、技術支援のオンライン窓口として、TAC Web サイトを提供しています。

TAC Web サイト

TAC Web サイト（http://www.cisco.com/tac）では、オンラインで資料やツールを利用して、トラブルシューティングやシスコ製品およびテクノロジーに関する技術上の問題の解決に役立てることができます。TAC Web サイトは 1 年中いつでも利用することができます。

TAC Web サイト上のツールにアクセスする際は、いずれも Cisco.com のログイン ID およびパスワードが必要です。サービス契約が有効で、ログイン ID またはパスワードを取得していない場合は、次の URL で登録手続きを行ってください。

http://tools.cisco.com/RPF/register/register.do

Japan TAC Web サイト

Japan TAC Web サイトでは、利用頻度の高い TAC Web サイト（http://www.cisco.com/tac）のドキュメントを日本語で提供しています。Japan TAC Web サイトには、次の URL からアクセスしてください。

http://www.cisco.com/jp/go/tac

サポート契約を結んでいない方は、「ゲスト」としてご登録いただくだけで、Japan TAC Web サイトのドキュメントにアクセスできます。

Japan TAC Web サイトにアクセスするには、Cisco.com のログイン ID とパスワードが必要です。ログイン ID とパスワードを取得していない場合は、次の URL にアクセスして登録手続きを行ってください。

http://www.cisco.com/jp/register/

TAC レベル情報の参照

オンライン TAC Case Open ツール（http://www.cisco.com/tac/caseopen）では、P3 および P4 の問題について最も迅速にテクニカル サポートを受けられます（ネットワークの障害が軽微である場合、あるいは製品情報が必要な場合）。状況をご説明いただくと、TAC Case Open ツールはすみやかな問題解決に役立つリソースを自動的に推奨します。

これらの推奨リソースを使用しても問題が解決しない場合は、TAC の技術者が対応します。

問題が P1 または P2（運用中のネットワークがダウンした場合、あるいは重大な障害が発生した場合）であるか、インターネットにアクセスできない場合は、電話で TAC にご連絡ください。P1 および P2 の問題には TAC の技術者がただちに対応し、業務を円滑に運営できるよう支援します。

電話でテクニカル サポートを受ける際は、次の番号のいずれかをご使用ください。

アジア太平洋：+61 2 8446 7411（オーストラリア：1 800 805 227）

EMEA：+32 2 704 55 55

米国：1 800 553-2447

TAC の連絡先一覧については、次の URL にアクセスしてください。

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

TAC プライオリティの定義

すべての問題を標準形式で報告するために、問題のプライオリティを定義しました。

- **プライオリティ 1（P1）**— ネットワークがダウンし、業務に致命的な損害が発生する場合。24 時間体制であらゆる手段を使用して問題の解決にあたります。
- **プライオリティ 2（P2）**— ネットワークのパフォーマンスが著しく低下、またはシスコ製品のパフォーマンス低下により業務に重大な影響がある場合。通常の業務時間内にフルタイムで問題の解決にあたります。
- **プライオリティ 3（P3）**— ネットワークのパフォーマンスが低下しているが、ほとんどの業務運用が機能している場合。通常の業務時間内にサービスの復旧を行います。
- **プライオリティ 4（P4）**— シスコ製品の機能、インストレーション、基本的なコンフィギュレーションについて、情報または支援が必要で、業務への影響がほとんどまたはまったくない場合。

概要

この章では、サービス コントロールの概念について全般的な概要を示します。SCE プラットフォームの機能的なトポロジーについて説明し、プラットフォームの動作とシステムにおける情報のフローについて全体像を示します。

SCE プラットフォームは、インターネット /IP トラフィックの観察、解析、および制御をサポートする目的で設計されました。SCE プラットフォームは、トラフィックの監視、解析、制御のためのさまざまな機能およびツールに生データを提供し、サービス プロバイダー（SP）の収益に貢献するとともに OPEX（運用経費）を抑えます。Service Control アプリケーションをサポートすることにより、SP による高度な差別化、Quality of Service（QoS; サービス品質）、マーケット セグメンテーションを実現可能にし、顧客の満足度を高めます。

この章の内容は次のとおりです。

Cisco サービス コントロールの概念

Cisco サービス コントロールの概念は、サービス コントロールに関してサービス プロバイダーが直面するさまざまな課題に対処するために設計されたハードウェアおよび固有のソフトウェア ソリューションの組み合わせによって実現されています。SCE プラットフォームは、インターネット/IP トラフィックの観察、解析、および制御をサポートする目的で設計されました。

サービス コントロールにより、サービス プロバイダーは既存のインフラストラクチャを利用しながら、収益性の高い新たな収入源を作ることができます。サービス コントロールの機能を活用することで、サービス プロバイダーは IP ネットワーク トラフィックをマルチギガビットのワイヤ速度で解析、課金、および制御することができます。Cisco サービス コントロール ソリューションでは、目標となる利益率の高いコンテンツ ベース サービスの見極めに必要なツールも提供されます。

通信業界の沈滞を見てもわかるとおり、IP サービス プロバイダーの収益性を確保するには、ビジネス モデルの改革が必要です。大規模データ リンクの構築に何十億ドルも投資してきたので、プロバイダーは多額の債務とコストの上昇に直面しています。それと同時に、アクセスと帯域幅がごく日常的なものになり、料金は低下し、結果的に収益が失われています。ネットワーク上のトラフィックとサービスからより大きな収益を引き出すには、付加価値サービスの提供が必要であることを、サービス プロバイダーは実感しています。ただし、IP サービスから実際に利益を生み出すには、データ リンク上でそのようなサービスを実行するだけでは不十分です。提供するサービスの詳細なモニタリングと精度、リアルタイムでの制御、および認識が要求されます。このようなギャップを埋めるのが、シスコが提供するサービス コントロール ソリューションです。

サービス コントロール機能

シスコのサービス コントロール プラットフォームの中心は、専用のネットワーク ハードウェア デバイスである Service Control Engine（SCE）です。完全なサービス コントロール ソリューションを実装するには、SCE で一定の機能を提供する必要があります。次に、Cisco SCE の中心的な機能を示します。これらの機能は、サービス コントロール ソリューションのための広範囲のアプリケーションをサポートします。

- サブスクライバ アウェアネスとアプリケーション アウェアネス：加入者（サブスクライバ）単位の使用状況およびコンテンツをリアルタイムで把握して制御するために、IP トラフィックをアプリケーション レベルで掘り下げます。
  - サブスクライバ　アウェアネス：IP フローと特定のサブスクライバを対応付け、プラットフォーム経由でトラフィックを伝送している各サブスクライバのステートを維持し、そのサブスクライバ トラフィックに適切なポリシーを実施します。

    サブスクライバ アウェアネスは、サブスクライバ管理用リポジトリ（DHCP サーバ、RADIUS サーバなど）との統合によって達成されます。
  - アプリケーション アウェアネス：アプリケーション プロトコル レイヤ（レイヤ 7）までトラフィックを認識および解析する能力。

    バンドルされたフローで実装されているアプリケーション プロトコル（たとえば、制御フローとデータ フローで実装される FTP など）に関して、SCE プラットフォームはフロー間のバンドル接続を認識し、それに応じた取り扱いをします。
- ステートフルなリアルタイムのトラフィック制御：ステートフルなリアルタイムのトラフィック トランザクション処理を利用して、きめ細かい帯域幅メータリングおよびシェーピング、クォータ管理およびリダイレクションなど、高度な制御機能を実行する能力。これには、適応力に優れたプロトコルとアプリケーション レベルのインテリジェンスが必要です。
- プログラマビリティ：絶えず変化するサービス プロバイダー環境で、すみやかに新しいプロトコルを追加し、新しいサービスやアプリケーションに簡単に適応する能力。プログラマビリティは、SML 言語によって達成されます。

  プログラマビリティは、新規サービスの迅速な展開に直結し、ネットワーク、アプリケーション、またはサービスの拡大のための簡易なアップグレード パスを提供します。

- 安定性のあるフレキシブルなバック オフィス統合：サービス プロバイダーの既存のサードパーティ製システム（プロビジョニング システム、サブスクライバ リポジトリ、課金システム、OSS システムなど）との統合能力。SCE には、ドキュメントを完備したオープンな API の集合があるので、迅速で安定性のある統合が可能です。
- スケーラブルで高性能なサービス エンジン：上記のすべてをワイヤ速度で実行する能力。

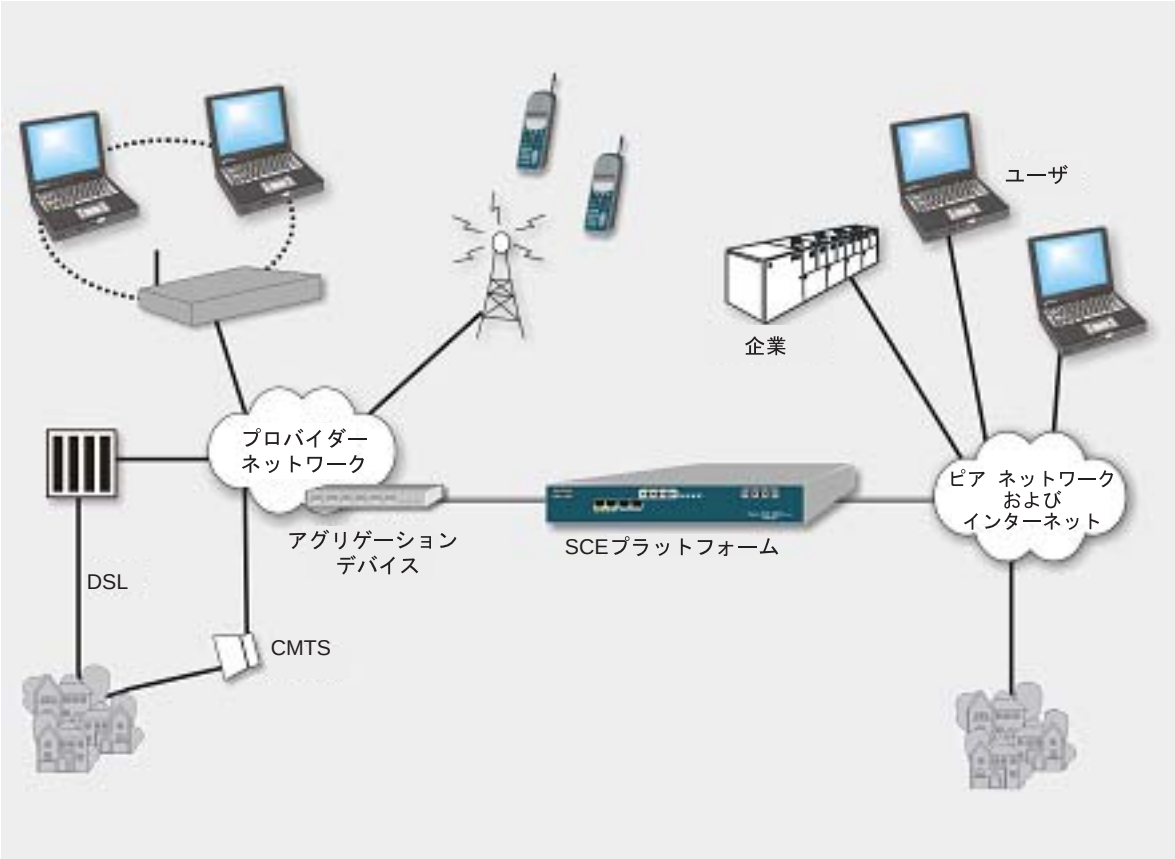
SCE プラットフォーム

SCE ファミリーのプログラマブル ネットワーク デバイスは、IP トラフィックのステートフル フロー インスペクションを実行し、設定可能なルールに基づいてトラフィックを制御することができます。SCE は、ASIC コンポーネントおよび RISC プロセッサを利用した専用ネットワーク デバイスであり、単なるパケット カウンティングにとどまらず、ネットワーク トラフィックの内容を深く調査します。双方向のトラフィック フローに関するプログラマブルなステートフル インスペクションと、これらのフローをユーザ オーナーシップに対応付ける能力により、SCE プラットフォームはネットワーク使用をリアルタイムで分類します。この情報が、SCE の高度なトラフィック制御および帯域幅シェーピング機能の基盤になります。ほとんどの帯域幅シェーパー機能はここまでですが、SCE はさらに多くの制御オプションおよびシェーピング オプションを提供しています。それは次のとおりです。

- レイヤ 7 ～ 3 のステートフルなワイヤ速度でのパケット インスペクションおよびパケット分類
- 600 以上のプロトコル / アプリケーションの安定したサポート：
  - 汎用：HTTP、HTTPS、FTP、TELNET、NNTP、SMTP、POP3、IMAP、WAP など
  - P2P：FastTrack-KazaA、Gnutella、WinMX、Winny、Hotline、eDonkey、DirectConnect、Piolet など
- ストリーミングとマルチメディア：RTSP、SIP、HTTP-STREAMING、RTP/RTCP など
- プログラマブルなシステム コアによるフレキシブルなレポート生成および帯域幅制御
- 既存のネットワークへのトランスペアレントなネットワークおよび BSS/OSS の統合
- サブスクライバ　アウェアネスによるトラフィックおよび使用状況の特定のカスタマーへの対応付け

次の図に、ネットワーク上での SCE プラットフォームの配置例を示します。

図 1-1　ネットワーク上の SCE プラットフォーム



サービス コントロール ソリューション

SCAS BB（SCAS）は、ブロードバンド サービス プロバイダーがネットワーク リソースの配分状態を把握して制御し、ビジネス戦略に適合するようにトラフィックを最適化するためのサービス コントロール ソリューションです。ネットワーク コストの削減、ネットワーク パフォーマンスおよびカスタマー エクスペリエンスの向上と、新しいサービス項目およびパッケージの作成を可能にします。

この章の内容は次のとおりです。

システム コンポーネント

Service Control Application Suite for Broadband ソリューションは、次の 3 つの主要コンポーネントで構成されています。

- SCE プラットフォーム：ネットワーク トランザクションをアプリケーション レベルで解析してレポートする目的で設計された、強力でフレキシブルな専用ネットワーク使用状況モニタ。

  SCE プラットフォームのインストレーションおよび運用については、『*SCE 1000 and SCE 2000 User Guide*』を参照してください。

- smartSUB Manager（SM）：サブスクライバ情報とサービス コンフィギュレーションのダイナミック バインディングが必要な場合に使用するミドルウェア ソフトウェア コンポーネント。SM はサブスクライバ情報を管理し、複数の SCE プラットフォームにリアルタイムでプロビジョニングします。SM はサブスクライバのサービス コンフィギュレーション情報を内部に保存し、AAA システム（RADIUS、DHCP など）と SCE プラットフォームの間のステートフル ブリッジとして動作します。

  smartSUB Manager のインストレーションおよび運用については、『*smartSUB Manager User Guide*』を参照してください。

- Collection Manager（CM）：1 台または複数の SCE プラットフォームからの RDR を待ち受ける収集システム。使用状況に関する情報および統計情報を収集し、バンドルされたデータベースにその情報を保存し、そのデータから各種の有益なレポートを提供します。また、CM はサブスクライバの使用状況に関する情報および統計情報をシンプルなテキスト ベースのファイルに変換し、これを外部システムでさらに処理したり収集したりすることもできます。

  Collection Manager のインストレーションおよび運用については、『*Collection Manager User Guide*』を参照してください。

SCE プラットフォーム、Collection Manager、および smartSUB Manager が連携して、IP ネットワーク トラフィックの詳細な観察、解析、レポート、および制御を実行します。Collection Manager および smartSUB Manager はオプションのコンポーネントであり、このソリューションのすべての展開に必須というわけではありません。サードパーティ製の収集アプリケーションおよびレポート生成アプリケーションを採用しているサイトや、サブスクライバ対応のダイナミックな処理を必要としないサイトでは、これらのコンポーネントは不要です。

次の図に、SCAS BB ソリューションにおける情報のフローを示します。

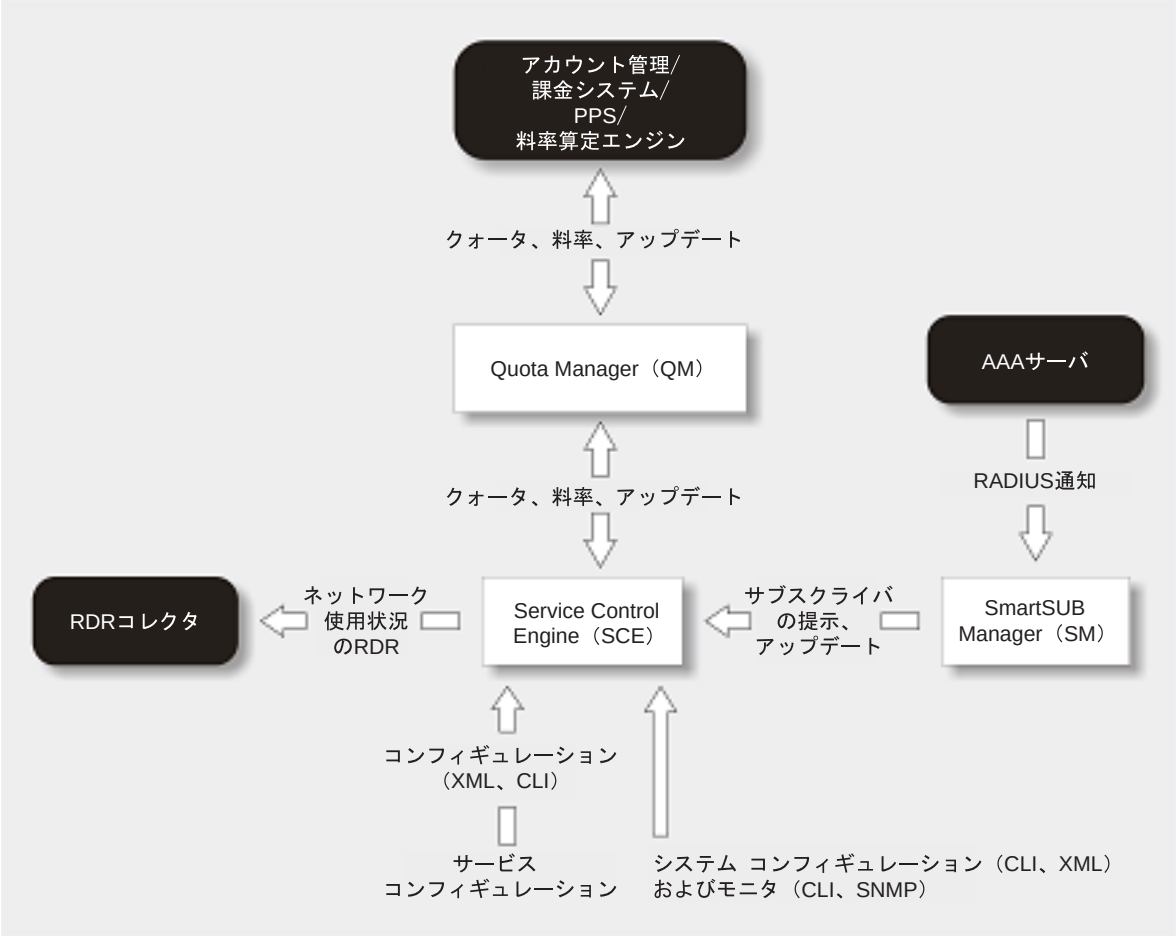
- 水平方向のフロー：サブスクライバと IP ネットワークの間のトラフィックを表します。

  SCE プラットフォームがトラフィック フローをモニタします。

- 垂直方向のフロー：SCE プラットフォームから Collection Manager への Raw Data Record（RDR）の送信を表します。

  smartSUB Manager を追加して、フローを制御しサブスクライバ データを提供することもできます。その場合、SCAS BB はダイナミックなサブスクライバ統合を処理できます。

図 2-1　SCAS システムの論理コンポーネント



サブスクライバおよびサブスクライバ モード

SCAS BB ソリューションの基本的なエンティティの 1 つが、*サブスクライバ*です。サブスクライバは、SCAS BB ソリューションで最も細かい粒度でサービス コンフィギュレーションを個別にモニタ、アカウンティング、および実施できるエンティティです。SCAS BB システムの最も粒度の細かいインスタンスであるサブスクライバは、それぞれ個別のサービス コンフィギュレーションを実施すべき、サービス プロバイダーの実際のカスタマーです。ただし、SCAS BB ソリューションでは、たとえばサブネット単位またはアグリゲーション デバイス単位でのトラフィックのモニタおよび制御のように、より大きい粒度でトラフィックをモニタおよび制御することもできます。

SCAS BB ソリューションの設計上、最も重要な決定事項の 1 つは、システムにおけるサブスクライバの定義です。この定義によって、使用するサブスクライバ モードが決まります。さらに、サブスクライバ モードによって（必要な場合）どのような統合が必要であるか、どのような実際のサービス コンフィギュレーションを定義すべきかが決まります。以下に、サポートされる各種のサブスクライバ モード、各モードでサポートされる機能、および前提条件と必要なコンポーネントについて説明します。

SCAS BB がサポートするサブスクライバ モードは、次のとおりです。

- **サブスクライバレス モード：**サブスクライバを定義しません。
- **アノニマス サブスクライバ モード：**IP アドレスを個別に制御およびモニタします。SCE プラットフォームは IP アドレスが使用されると自動的にそのアドレスを認識し、デフォルトのサービス コンフィギュレーションを割り当てます。
- **サブスクライバアウェア モード ― スタティック サブスクライバ：**着信 IP アドレスは、システム オペレータが設定したサブスクライバにスタティックにバインドおよびグループ化されます。
- **サブスクライバアウェア モード ― ダイナミック サブスクライバ：**サブスクライバに IP アドレスを割り当てるシステム（RADIUS、DHCP）との統合により、サブスクライバが現在使用している IP アドレスにサブスクライバ情報がダイナミックにバインドされます。サービス コンフィギュレーション情報は、直接サービス コントロール ソリューションで管理するか、または統合によってダイナミックにプロビジョニングします。

サブスクライバレス モード

サブスクライバレス モードは、グローバルなデバイス レベルのみで制御機能およびレベル解析機能が必要とされるサイトで使用します。たとえば、リンク上の P2P トラフィックの総量をモニタする目的で使用できます。

サブスクライバレス モードでは統合が不要なので、smartSUB Manager コンポーネントは必要ありません。サブスクライバレス モードはサブスクライバ数や着信 IP アドレス数に影響されないので、SCE プラットフォームの視点では、モニタ対象のリンクを利用するサブスクライバの総数に制限はありません。

アノニマス サブスクライバ モード

アノニマス サブスクライバ モードは、サブスクライバ着信 IP アドレスの粒度でネットワーク トラフィックを解析して制御するための手段を提供します。このモードは、サブスクライバごとに差別化された制御やサブスクライバ別のクォータ トラッキングが不要な場合、IP レベルでの解析で十分である場合、または IP アドレス / サブスクライバのバインディングをオフラインで実行できる場合に使用します。たとえば、どのサブスクライバが最大の P2P トラフィックを発生させているかを調べるには、トップ IP アドレスを識別し、それらのアドレスを RADIUS/DHCP ログから手動 / オフラインで個々のサブスクライバに対応付けます。各サブスクライバが使用できる P2P トラフィックの合計帯域幅を制限することも可能です。

アノニマス モードでは、統合が不要で、使用する IP アドレスのスタティックな設定も不要なので、smartSUB Manager コンポーネントは必要ありません。このモードでは、直接 SCE プラットフォームに IP アドレスの範囲を設定します。システムはこの範囲で IP アドレスをサブスクライバ名として使用し、「アノニマス」サブスクライバをダイナミックに作成します。SCE プラットフォームがサポートする、同時にアクティブなアノニマス サブスクライバの総数は、同時にアクティブなサブスクライバの総数と同じです。

スタティック サブスクライバ モード

スタティック サブスクライバ モードは、着信 IP アドレスをまとめてグループにバインドし、定義済みサブスクライバとの間のトラフィックを 1 つのグループとして制御できるようにします。たとえば、このモードを使用する場合、（複数のサブスクライバが同時に使用する）特定のネットワークのサブネットとの間のすべてのトラフィックを「仮想サブスクライバ」として定義し、1 つのグループとして制御および表示することができます。

スタティック サブスクライバ モードは、サービス コントロール ソリューションで制御するエンティティが、ダイナミックに変更されることのない一定の IP アドレスまたはアドレス範囲を使用する場合に対応します。具体的には次のような場合です。

- サブスクライバの IP アドレスが DHCP、RADIUS などでダイナミックに変更されない環境。
- 1 つのサブスクライバ グループが共通の IP アドレス プールを使用する場合。たとえば、特定の CMTS、BRAS などでサービスするすべてのサブスクライバを一括して管理し、グループ全体で帯域幅を共有させる場合など。

スタティック サブスクライバは直接 SCE プラットフォームに定義することができ、外部の管理ソフトウェア（smartSUB Manager）は不要です。SCE プラットフォームの CLI を使用して、サブスクライバのリスト、IP アドレス、および対応するパッケージを定義します。

サブスクライバアウェア モード — ダイナミック サブスクライバ

ダイナミック サブスクライバ モードでは、サブスクライバが現在使用している（IP）アドレスにダイナミックにバインドされたサブスクライバ情報（OSS ID およびサービス コンフィギュレーション）が SCE に入力されます。その結果、使用されている IP アドレスとは無関係に、サブスクライバ単位で差別化されたダイナミックな制御と、サブスクライバ レベルでの解析が可能になります。このモードは、サブスクライバ レベルでトラフィックを制御および解析し、IP アドレスとは無関係にサブスクライバの使用状況をモニタする場合に使用します。また、サブスクライバごとに異なるサービス コンフィギュレーションまたはパッケージの割り当てと実施も可能です。

このモードでは、smartSUB Manager（SM）を使用してデバイスにサブスクライバ情報をプロビジョニングする必要があります。SM は、上記のような関連付けを維持するサーバ アプリケーションであり、この情報を SCE プラットフォームにリアルタイムでプロビジョニングします。

サブスクライバ モード ― 要約

次の表に、サポートされる各種のサブスクライバ モードを要約します。

表 2-1　サブスクライバ モードの要約

モード	サポートする機能	主な利点	使用する状況
サブスクライバレス	グローバル（デバイス レベル）での解析および制御	サブスクライバの設定が不要	グローバルな制御ソリューションまたはサブスクライバ レベルでの解析。例： • ピアリング ポイントでの P2P アップロードを制御する • P2P の合計量を固定的なパーセンテージに制限する
アノニマス サブスクライバ	グローバルでの解析および制御 個々のIPアドレス レベルでの解析および制御	サブスクライバの設定が不要 使用するサブスクライバ IP アドレス範囲の定義のみ必要 統合を必要としないサブスクライバ レベルでの制御	サブスクライバ単位での差別化のない IP レベルの解析または制御を行う場合、およびオフラインでの IP アドレス / サブスクライバ バインディングで十分な場合。例： • サブスクライバ 1 人あたりの P2P を 64 Kbps（キロビット / 秒）に制限する • トップ サブスクライバを識別する（トップ IP アドレスを識別し、RADIUS/DHCP ログと手動 / オフラインで照合する）
スタティック サブスクライバ	グローバルでの解析および制御 SCE プラットフォームでスタティックに設定した個別のIPアドレス/グループに基づく制御	統合を必要としない、定常的でスタティックなサブスクライバ設定 サブスクライバ　トラフィックを論理グループで管理	サブスクライバのグループのトラフィックを制御する場合。例： • 特定の CMTS デバイスを使用するサブスクライバのグループごとに、P2P トラフィックを 5 Mbps（メガビット / 秒）に制限する
ダイナミック サブスクライバ	すべてのシステム機能	サブスクライバ単位での差別化されたダイナミックな制御 使用されている IP アドレスとは無関係な、サブスクライバ レベルでの解析	次のような場合： • サブスクライバ レベルでトラフィックを制御および解析する • IP アドレスとは無関係にサブスクライバの使用状況をモニタする • サブスクライバ別に異なるサービス コンフィギュレーションまたはパッケージを割り当て、ダイナミックにパッケージを変更する

サービス コンフィギュレーション

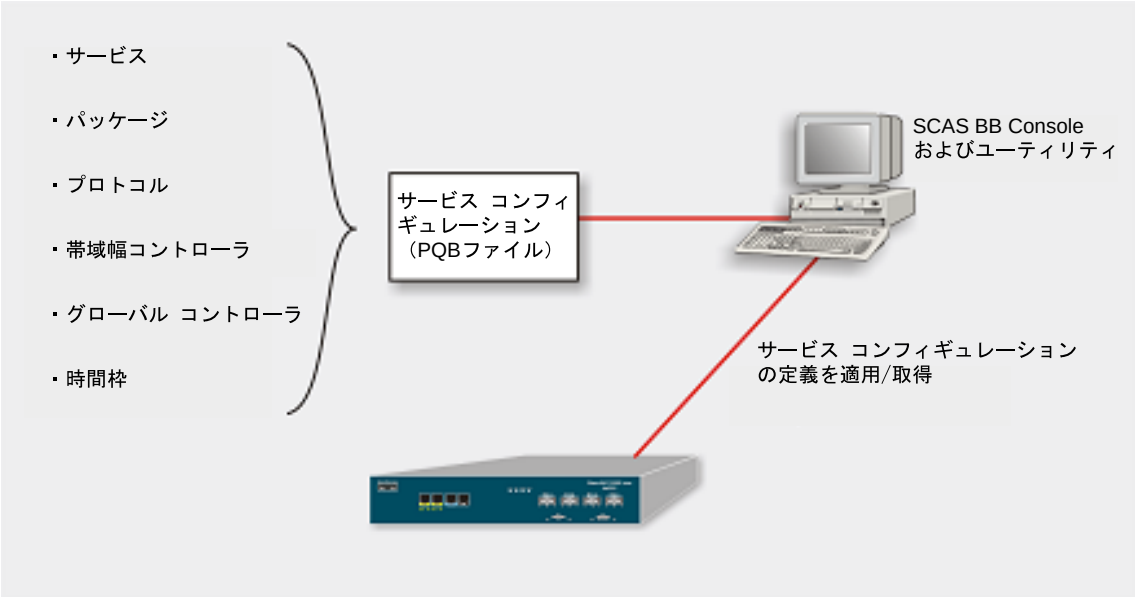
サービス コンフィギュレーションは、SCE プラットフォームによるトラフィックの解析方法および制御方法を定義します。大まかに言うと、サービス コンフィギュレーションは次のものを定義します。

- プロトコルおよびサービス分類
- パッケージおよびサービス コンフィギュレーション
- 帯域幅コントローラ
- グローバル コントローラ

サービス コンフィギュレーションは、次のいずれかを使用して行います。

- SCAS BB Console
- Service Configuration ユーティリティ
- SCAS BB API

図 2-2　サービス コンフィギュレーション



SCAS BB Console

SCAS BB Console は、サービス コンフィギュレーションの作成、変更、適用に使用する SCAS BB GUI です。SCAS BB Console を使用して、サービス、パッケージ、プロトコル、帯域幅制御などの設定エンティティを定義できます。SCAS BB Console で作成したサービス コンフィギュレーション ファイル（*.pqb*）は、SCE プラットフォームに保存または適用できます。

また、SCAS BB Console から smartSUB Manager にアクセスしてサブスクライバ管理を行うこともできます。さらに、Collector Manager の Reporter 機能にアクセスしてレポートを作成し出力することもできます。

SCAS BB Console についての詳細は、『*SCAS BB User Guide*』を参照してください。

Service Configuration ユーティリティ

Service Configuration ユーティリティは、**.pqb** コンフィギュレーション ファイルを SCE プラットフォームに適用したり、SCE プラットフォームから現在のコンフィギュレーションを取得して **pqb** ファイルとして保存するための、簡素なコマンドライン ツールです。このツールは、Windows または Solaris 環境にインストールして実行できます。**.pqb** ファイルのサービス コンフィギュレーションを使用して SCE プラットフォームを設定できます。

Service Configuration ユーティリティ（`servconf`）は、サービス コンフィギュレーションを適用および取得するためのコマンドライン インターフェイスを提供し、これらの動作を自動的に実行します。

Service Configuration ユーティリティについての詳細は、『*SCAS BB User Guide*』を参照してください。

SCAS BB サービス コンフィギュレーション API

SCAS BB Service Configuration API は、サービス コンフィギュレーションのプログラミングと管理、および SCE プラットフォームへの適用を行うための Java クラスの集合です。また、SCAS API を使用するアプリケーションをサードパーティ製システムに統合し、サービス プロバイダーによる管理および運用タスクを簡易化することもできます。

CHAPTER 3

開発者のためのシステム アーキテクチャ

この章の目的は次のとおりです。

- SCAS BB システムのアーキテクチャを SCAS 開発者向けに説明します。
- SCAS BB の主要なハードウェアおよびソフトウェア コンポーネントを紹介します。
- SCAS API によるプログラミングの定義と概念を示します。
- SCAS BB の各ライセンス レベルについて説明します。
- SCAS BB システムの各種の論理エンティティ間の接続について説明します。
- SCAS BB ハードウェアおよびソフトウェア コンポーネント間での情報のフローについて説明します。

SCAS BB の Java API クラスは、SP による付加価値インターネット サービスおよびネットワーク トラフィック モニタリング アプリケーション開発のための優れたプログラミング環境を提供します。

この章の内容は次のとおりです。

統合の必要性と意義

SCAS API を使用すると、サービス プロバイダーは次のことを実現できます。

- **より大規模な SP ソリューションの一部への SCAS BB の組み込み。**大規模な SP ソリューションで、部分的に重複したオペレーションが存在していたり、類似した複数のシステムをメンテナンスしたりすることは許されません。AAA や課金といった既存のシステムに、よりコンパクトかつネイティブに、トランスペアレントに統合する必要があります。
- **新たな収益源によるビジネスの拡大。**API のプログラミング機能を利用して、高度にカスタマイズされたアプリケーションおよびソリューションを実現できます。SP のインハウス システム用アプリケーションでも、一般向けのソフトウェア アプリケーションやサービスでも開発できます。たとえば、カスタマーが好みのサービス パッケージを選択できる Web インターフェイスを開発する場合、このようなアプリケーションを作成するためのソフトウェア プラットフォームおよびプログラミング ツールが API によって SP に提供されます。
- **運用タスクの簡易化**
- **レガシー システム GUI のルック アンド フィールの保持**

さらに、SCAS BB システムをプロバイダー ネットワークに統合する場合、既存のハードウェアやインフラストラクチャに手を加えたり、これらを廃止したりする必要はありません。SCAS BB システムはハードウェアおよびインストール済みソフトウェアとともに提供され、最低限の設定作業だけでプロバイダー ネットワークに配置できます。

基本コンポーネント

管理 LAN 上で動作する SCAS BB システム コンポーネントは、ネットワーク サブスクライバとインターネットの中間に位置します。

SCAS BB システムがネットワーク フローに対して実行する主な機能は、次の 3 つです。

- **ネットワークの解析** ― ネットワーク トラフィックを解析し、ネットワークを使用しているトランザクションのタイプを判別します。
- **サブスクライバの解析** ― ネットワークでトランザクションを実行しているサブスクライバまたは組織の特性を解析します。
- **サービス コンフィギュレーションの実施** ― トラフィック解析に基づいて、ネットワーク上でサービス コンフィギュレーションを実施します。ネットワーク サービスのプロビジョニング、サブスクライバの差別化を可能にします。

SCAS BB システムの中心的なコンポーネントは次のとおりです。

- SCE プラットフォーム（SCE 1000 および SCE 2000）
- smartSUB Manager（SM）
- （オプション）1 つまたは複数の Collection Manager システム（CM）
- SCAS クライアント（SCAS BB Console、SCAS smartSUB Manager、SCAS Reporter）

SCE プラットフォーム

SCE プラットフォームは、サービス プロバイダーおよびバックボーン キャリアのネットワーク機能を拡張する目的で設計された、専用のサービス コンポーネントかつアクティブな実施システムです。SCE プラットフォームは、複雑なトラフィック フローをワイヤ速度で識別、分類、処理することにより、単なるトランスポート ネットワークを、広い範囲の付加価値 IP アプリケーション（ビデオ ストリーミング、VoIP、ティアード サービス、2 社間アプリケーション レベル SLA など）に対応する差別化されたサービス配信インフラストラクチャへ進化させます。

SCE プラットフォームは業界標準のインターフェイスおよび通信プロトコルを使用して、既存のネットワーク エレメント（ルータ、スイッチ、アグリゲータ、サブスクライバ管理デバイス、運用サポート システムなど）とシームレスに連携します。

ネットワークを通過するパケットが、着信する速度で処理されることを保証するには、カスタムメイドのハードウェア ソリューションが必要になります。

SCE プラットフォームには、3 つのモデル（SCE 1000、SCE 2000 4xGBE、および SCE 2000 4/8xFE）があります。1 台または複数の SCE プラットフォームをプロバイダー ネットワークで使用できます。SCE プラットフォームの内部では、ネットワーク トランザクションが解析され、プロバイダーのポリシーを実施するサービスに対応付けられます。

さらに、SCE プラットフォームは、システム ソリューションのビジネス ロジックを実装し、トランザクション解析をリアルタイムで実行します。SCE プラットフォームの設定により、Raw Data Record（RDR）をシステムのデータ リポジトリである Collection Manager（CM）に送信して保存したり、その他の処理（帯域幅の制御、容量の制御など）を実行することもできます。

smartSUB Manager

smartSUB Manager（SM）は、SCE プラットフォームをサブスクライバアウェア モードで運用する場合に、サブスクライバの取り扱いに関する次の 3 つの問題に対処するためのソフトウェア ソリューションです。

- キャパシティ：SCE プラットフォームは（時間の経過とともに）スタティックに処理できる量を超えたサブスクライバを処理しなければならなくなる可能性があります。

  SM は、サブスクライバ名およびサブスクライバ情報を保管するリポジトリです。
- マッピング：SCE プラットフォームがネットワーク ID（IP アドレス）のあるフローを検出した場合、これらのネットワーク ID とサブスクライバ ID のマッピングが必要になります。

  SM データベースには、サブスクライバに対応するネットワーク ID が含まれています。
- ロケーション：どの SCE プラットフォームが、どのサブスクライバ トラフィックを処理するかを、システムが予測できない場合があります。

  SM により、サブスクライバを pull モードで導入し、実行時にどの SCE プラットフォームがどのサブスクライバを処理しているかを検知するようにシステムを設定できます。

SM データベースは、次のいずれかの方法で使用できます。

- 唯一のサブスクライバ情報源として：SM をスタンドアロン モードで使用する場合
- サブスクライバ情報キャッシュとして：SCE デバイスのグループとカスタマーの AAA および OSS システム間のブリッジとして SM を使用する場合

Collection Manager

Collection Manager（CM）は、SCE プラットフォームから Raw Data Record（RDR）などの使用状況レコードを受信して処理するソフトウェア コンポーネントです。CM は RDR を処理し、データベース、CSV（カンマ区切り）フラット ファイルなどのストレージ デバイスに保存のため送信します。一般的なシステム統合では、データ コレクタへの定期的なアクセスと、保存されている使用状況ファイルの処理を行います。

CM は、サーバ プラットフォーム上でレコードの送信先である Sybase データベースと併存させることもできますし、別のプラットフォームに配置することもできます。Sybase データベースのスクリプト例は、『*Collection Manager User Guide*』を参照してください。

SCAS BB ライセンス

SCAS BB では、さまざまなサイトのニーズに対応するため、次の 3 レベルのライセンスを用意しています。

- **SCAS BB View**：SCAS BB の基本形式であり、次の機能があります。
  - モニタおよびレポート
  - 制御機能なし
- **SCAS BB Capacity Control**：このライセンスでは、トラフィック制御機能が追加されます。次の機能があります。
  - モニタおよびレポート
  - キャパシティ制御（たとえば、各アプリケーションのトラフィックを異なるグローバル コントローラに割り当てる）
  - 単一のパッケージ（デフォルト パッケージのみ）― サブスクライバ間での差別化なし
  - キーが必要
- **SCAS BB Tiered Control**：このライセンスでは、パッケージに基づく差別化のあるトラフィック フロー制御が可能です。次の機能があります。
  - モニタおよびレポート
  - キャパシティ制御
  - 複数のパッケージ ― サブスクライバ間の差別化（たとえば、特定のパッケージの加入者には、広い帯域幅または 1 日あたりの大きい容量クォータを与える）
  - キーが必要

よりレベルの高いライセンスを登録する手順は、次のとおりです。

ステップ 1　**Help** メニューから **License Manager** をクリックします。

License Manager ダイアログが表示されます。

ステップ 2　**Enter new license key** チェックボックスをオンにします。

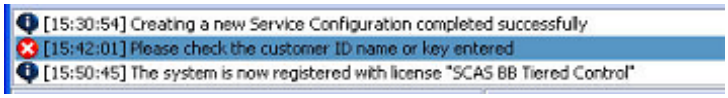
Customer ID および **Key** フィールドが使用可能になります。

ステップ 3　カスタマー ID およびキーを該当するフィールドに入力します。

ステップ 4　**OK** をクリックします。

メッセージ バンドとステータス バーに、新しいライセンスが表示されます。

図 3-1　新しいライセンスの表示



統合ポイント

SCAS BB ソリューションには、次のような統合シナリオがあります。

- **SM API** ― SM API でアプリケーションを統合すると、サブスクライバのプロビジョニング、パッケージへの割り当て、およびサブスクライバのダイナミックなパッケージ変更が可能になります。
- **Quota Provisioning API** ― QP API でアプリケーションを統合すると、サブスクライバに対するクォータのプロビジョニング、およびサブスクライバのクォータ残量の照会が可能になります。
- **Service Configuration API** ― Service Configuration API でアプリケーションを統合すると、リストの更新、サービス コンフィギュレーションの編集、および構成スクリプトを実行できます。
- **Reporter CLI** ― SCAS Reporter CLI（コマンドライン インターフェイス）でアプリケーションを統合すると、SQL データベースに定期的にアクセスしてバッチ クエリを実行したり、サードパーティ製の課金システムまたは統計解析パッケージで使用するイメージ ファイルを生成したりできます。
- **フラット ファイル** ― CSV（カンマ区切り）フラット ファイルを使用して、サードパーティ製の課金システムまたは統計解析パッケージにデータを統合できます。

Service Configuration API

Service Configuration API は、サービス コンフィギュレーションのプログラミングおよび管理を行うとともに、サービス コンフィギュレーションによってビジネス ルールを実施する SCE プラットフォームの機能を設定するための Java クラスの集合です。

Service Configuration API を使用して、Service Configuration Management モジュールで提供される機能を拡張した GUI アプリケーションを開発できます。Service Configuration API で作成したサーバ アプリケーションを通じて、SP のネットワーク ソリューションの展開およびカスタマイズされたアプリケーションの提供が可能になり、カスタマーのニーズを満たす付加価値サービスを提供できます。

Service Configuration API によるアプリケーションは、上記のような利点に加えて、サードパーティ製システムに統合可能です。

SCAS Reporter CLI

SCAS Reporter CLI は、Service Control Reporter GUI クライアントに相当する SCAS BB システム ソフトウェアです。Collection Manager に SQL クエリを実行し、出力をフィルタして、表またはグラフ形式のネットワーク トランザクション使用状況レポートを生成します。

SCAS Reporter CLI は、コマンドラインから独立型プログラムとして実行することも、各種のアプリケーションから起動することもできます。さらに、SCAS Reporter CLI は Service Control Reporter GUI クライアントの全機能を提供し、各種のシステムに統合して使用することができます。

フラット ファイル

CSV（カンマ区切り）フラット ファイルを使用するアプリケーションでは、データを表示して使用するとき、独自のフォーマットおよびテンプレートを適用できます。たとえば、Excel などのアプリケーションでは、CSV ファイルをワーク シートにインポートできます。

情報フロー

ここでは、SCAS BB の各種ハードウェアおよびソフトウェア コンポーネント間における情報フローについて説明します。SCAS BB システムは、1 台または複数の SCE プラットフォームと、SM、Collection Manager、その他の SCAS モジュールといった、相互にサービスを提供しあうソフトウェア コンポーネントで構成されます。各コンポーネントの関係と、システム内でデータが流れる方向を理解することで、開発者はシステムの全体像を把握できます。

SCAS BB システム コンポーネント間の重要な関係を、次に要約します。

- **サービス コンフィギュレーションの取得** ― SCE デバイスに存在するアクティブなサービス コンフィギュレーションのコピーを取得します。

  通常、サービス コンフィギュレーションを取得する目的は、サービス コンフィギュレーションの編集です。プロバイダーのビジネス ルールの変更に伴うスタティック プロセスです。
- **サービス コンフィギュレーションの適用** ― Service Configuration API の *apply* ファンクション コールを実行するアプリケーションが、SCE デバイスのアクティブなサービス コンフィギュレーションに代わる、サービス コンフィギュレーション変更情報を提供します。サービス コンフィギュレーションが転送されると、そのサービス コンフィギュレーションが有効になり、プロバイダーの新しいビジネス ルールがアクティブ化して、ネットワーク トラフィックに対して実施されます。
- **SCE プラットフォームによる RDR のトリガー** ― サブスクライバのネットワーク トランザクションに関するシステム定義の条件が満たされた場合にトリガーされるアクションの結果として、SCE プラットフォームが RDR を作成します。RDR に含まれる情報は、SCE プラットフォーム（作成元）から Collection Manager に送信されます。Collection Manager の設定に応じて、Collection Manager Adapter が、RDR をデータベース テーブルまたは CSV フラット ファイルに入力するための準備を行います。
- **SCAS Reporter の実行** ― 対話形式または Reporter CLI により、SCAS Reporter を使用してレポート機能が実行されると、ODBC ドライバを介して Sybase データベースに SQL クエリが実行されます。SQL クエリの結果は、実行中のレポートで指定のフォーマットに変換され、表示したりファイルに保存したりできます。

CHAPTER 4

サービス コンフィギュレーション エンティティ

この章の目的は次のとおりです。

- システムの技術的な概要を示します。
- SCAS API によるプログラミングの定義と概念を示します。

この章では、SCAS BB システムの主な要素（サービス コンフィギュレーション エンティティ）の定義を示します。これらは SCAS BB Java アプリケーションの構成単位です。したがって、各種のプログラマブル オブジェクト（サブスクライバ、サービス コンフィギュレーション、パッケージ、サービス、およびルール）により SCAS API の機能を最大限に活用してアプリケーションを開発するには、これらの要素の機能を理解しておくことが不可欠です。

この章の内容は次のとおりです。

論理エンティティ

ここでは、SCAS BB システムを構成する論理エンティティについて、プログラミングの観点から説明します。SCAS Java API はこれらの定義に基づいて、安定性のあるクライアント / サーバ アプリケーションを構築します。開発したアプリケーションは、ネットワークで実行することも、独立型アプリケーションとして使用することもできます。また、インターネットで配信されるサードパーティ製システムに統合することも可能です。

ここで説明するエンティティは次のとおりです。

これらのエンティティは、相互に密接な関係があります。したがって、あるエンティティについて詳しく説明する前に、そのエンティティが文中で言及される場合があります。

サービス コンフィギュレーション

サービス コンフィギュレーションは、サブスクライバの個人のニーズやビジネス ニーズに応じて選べるさまざまなパッケージを提供することにより、プロバイダーのビジネス戦略とビジョンを具体化します。パッケージは Service Configuration Editor で作成することも、ネットワーク プロバイダーの要件に応じて Service Configuration API でカスタムに作成することもできます。システムには、サービス コンフィギュレーションごとに最大で 64 個のパッケージ、31 個のサービスという制限があります。

サービス コンフィギュレーションを使用するには、適切な SCE プラットフォームにサービス コンフィギュレーションを伝播し、そのサービス コンフィギュレーションを実施およびアクティブ化する必要があります。SCE プラットフォーム上のアクティブなサービス コンフィギュレーションが、通過するネットワーク トラフィックを解析することによって、サービス コンフィギュレーションを実施します。

サービス コンフィギュレーションの構成要素は次のとおりです。

- **サービス** ― サービス コンフィギュレーションには、Web 閲覧、ビデオ ストリーミング、ビデオ会議などの *サービス*が含まれます。各サービスは、ネットワーク アクティビティをそのサービスに対応付ける方法を定義したトランザクション マッピングで構成されます。
- **パッケージ** ― サービス コンフィギュレーションには *パッケージ*が含まれます。各パッケージは、*サービス*別に定義されたルール（帯域幅レート制限、容量制限、アドミッション制御など）の集合で成り立っています。パッケージを構成するルールの集合ごとに、異なるサブスクライバ集団をターゲットとした完全なソリューションにする必要があります。

サービスおよびサービス エレメント

プロバイダーの視点では、サービスは顧客に販売するネットワーク製品です。プログラマの視点では、サービスは 1 つまたは複数のサービス エレメントで構成され、各エレメントは、ネットワーク トラフィックのトランザクション タイプに対応するサービスに関して一定の判断を可能にします。

SCE プラットフォームは上記のようなトランザクション マッピングを使用して、プラットフォームを通過するネットワーク接続を、それぞれ特定のサービスに対応付けます。その後、サービス別にルールを適用して、サービス コンフィギュレーションを実装します。分類ルールには、L3/4 パラメータ（ポート番号、IP アドレスなど）のほかに、L7 パラメータ（ホスト名、HTTP 接続のユーザ エージェントなど）を使用できます。

次の表に、サービスおよび対応するネットワーク パラメータの例を示します。

表 4-1　サービスおよびサービス パラメータの例

サービス名	プロトコル	起動側	アドレス リスト
Web 閲覧	HTTP HTTPS FTP	サブスクライバ起動	なし
Web ホスティング	HTTP HTTPS	ネットワーク起動	なし
ローカル ストリーミング	RTSP MMS	サブスクライバ起動	215.53.64.43 213.53.64.53

SCAS BB では、600 以上のプロトコルがサポートされています。これらのプロトコルを目的別に編成し、適切な名前（Web 閲覧、Web ホスティング、ビデオ会議、ビデオ ストリーミング、ローカル コンテンツなど、サブスクライバ向けの購入パッケージの一部分に使用できる名前）を付けると、あとでマーケティング サービスに役立ちます。

次にサービスの例を示します。

- **Web 閲覧** ― 発信 HTTP トランザクションに対し、インターネット Web 閲覧をイネーブルにするか制限します。
- **Web ホスティング** ― 着信 HTTP トランザクションに対し、インターネット Web ホスティングを提供します。
- **ビデオ会議** ― RTSP トランザクションに対し、ビデオ会議機能を提供します。
- **ゲーム サービス** ― 無制限の帯域幅で、特定のネットワーク サービス（ゲーム アプリケーションなど）へのアクセスを提供します。
- **ローカル コンテンツ ゲーム** ― ゲーム サービスを格安な料金でサブスクライバに提供することによって、ゲーム アプリケーション（プロバイダーのローカル ネットワークの外部にあるサーバではなく、プロバイダー ネットワーク内のローカル サーバでホスティングされる Doom など）の利用を促します。
- **株価情報サービス** ― 投資家に人気のある株価情報サービスに課金します。

SCAS BB のオブジェクト指向型テクノロジーを使用すると、サービス オブジェクトなどのプログラム要素を同じ名前で定義し、その機能をネットワーク型のシステム レベル アプリケーションにシームレスに統合することができます。プログラマブルなサービスの利点の 1 つは、ネットワーク サービスの配信方法を強力に制御できる点です。

複数のサービス エレメントで構成されるサービスの一例として、あるゲーム サービスでは、1 つのサービス エレメントがポート 666 のプログラム Doom に対応するネットワーク分類エンティティを定義し、もう 1 つのサービス エレメントがポート 333 のプログラム Quake に対応する分類エンティティを定義します。ゲーム アプリケーションは大量の帯域幅を消費するので、サブスクライバがサービスに接続している間に使用した帯域幅容量を測定し、それに応じて課金することが、このゲーム サービスに適したサービス ルールとなります。

ルールは、サービスを構成するサービス エレメントに対してではなく、サービスに対して定義します。したがって、システムをプランニングするときは、一連のサービス エレメントを共通のルールベース機能および特性と一括して指定することが、設計上の重要な考慮事項となります。

実務上、システム サービスは一連のサービス エレメントで構成されます。次に、サービス エレメントの主なコンポーネントを示します。

- **プロトコル** ― サービス エレメントに対応付けるプロトコルの名前。
- **起動側** ― トランザクションを伝送する方向に制限があるかどうかを表します。方向としては、ネットワーク起動、サブスクライバ起動、またはその両方があります。
- **（オプション）リスト** ― リストで指定した IP アドレスまたはホスト名に、トランザクションを対応付けるかどうかを表します。
- 上記のサービス エレメントの各コンポーネントについて、以下の各項で説明します。

プロトコル

トランザクションのプロトコルは、ポート番号およびトランスポート タイプによって判別されます。これらのパラメータに基づいて、ネットワーク トランザクションに関する詳細な解析（レイヤ 7）が実行されます。たとえば、ポート番号が 80、トランスポート タイプが TCP で、コンテンツがプロトコルのシグニチャに一致する場合、システムのテーブルがチェックされ、そのトランザクションは HTTP プロトコルに対応付けられます。ポート番号が 1755、トランスポート タイプが UDP で、コンテンツがプロトコルのシグニチャに一致する場合、そのトランザクションは RTSP（ビデオ ストリーミングに使用するプロトコル）に対応付けられます。

SCAS BB の今後のバージョンでは、サービス マッピング方式に使用される既存のネットワーク トランザクションの範囲が広がる予定です。これらの新しいマッピング方式の応用例として、たとえば電子メールの件名に「I love you」というテキストが含まれているものを弾き出し、ウィルス スキャニング アプリケーションで処理してから、サブスクライバに配信することが可能になります。その場合、コンピュータ ウィルスをプロバイダーまたはサブスクライバのコンピュータ システムに着信する前にトラップして、被害を未然に防ぐことができます。一方、これらの新しいマッピング方式を利用して、ウィルスがインターネットに到達するのを防ぐこともできます。SCAS BB システムは、着信トラフィックと発信トラフィックの両方を解析できるからです。

次に、SCAS BB システムにおけるプロトコルの主要事項をまとめます。

- システムの定義上、プロトコルとは、ポート番号とトランポート タイプの組み合わせです。
- サービス コンフィギュレーションにはプロトコルのリストがあり、各プロトコルのプロトコル名およびトランスポート タイプのほかに、UDP または TCP プロトコルの場合はポート番号、または IP プロトコル番号を指定します。ネットワーク トランザクションのプロトコルは、これらのパラメータによって識別されます。
- サービス コンフィギュレーションのいずれかのプロトコルで、TCP または UDP トランザクションのポート番号が定義されていない場合、そのトランザクションのプロトコルは「汎用 TCP」または「汎用 UDP」プロトコルとして識別されます。
- サービス コンフィギュレーションのいずれかのプロトコルで、IP プロトコル番号が定義されていない場合、そのトランザクションのプロトコルは「汎用 IP」プロトコルとして識別されます。

ダイナミック シグニチャ

ダイナミック シグニチャは、コンフィギュレーションに新しいプロトコルの分類を追加するためのメカニズムです。これは、新しいプロトコルがリリースされ、カスタマーがそのトラフィックを分類できるようにしたい場合に役立ちます（たとえば、P2P-Control ソリューションにおける新しい P2P プロトコルなど）。ダイナミック シグニチャは、特殊な Dynamic Signature Script（DSS）ファイルで提供されます。このファイルは、SCAS BB Console または API によって PQB ファイルに追加できます。PQB に DSS ファイルを読み込むと、サポートされる新しいプロトコルがプロトコル リストで使用可能になり、必要に応じてサービスに追加できるようになり、レポートを表示するときに使用されます。DSS ファイルは、カスタマーの要件およびマーケット ニーズに応じて、シスコまたはパートナーが定期的にリリースしています。

起動側

ネットワーク トランザクションの起動側は、ネットワーク起動（Network-Initiated）またはサブスクライバ起動（Subscriber-Initiated）のいずれかです。サブスクライバ起動のトランザクションは、サブスクライバからネットワークへ発信されるトランザクションです。一方、ネットワーク起動のトランザクションは、ネットワークからサブスクライバへ発信されるトランザクションです。

サービス エレメントで定義するネットワーク トランザクションを、(a) ネットワーク起動の方向、(b) サブスクライバ起動の方向に制限することもできますし、(c) どちらの方向にも制限しないことも可能です。

「方向」プロセスの作用を理解するため、いくつかのプロトコルについて次に検証します。

- ICQ プロトコルの場合、トランザクションを起動する方向はどちらでも構いません。インスタント メッセージには着信（Network-Initiated）および発信（Subscriber-Initiated）の両方が必要だからです。
- HTTP の場合、トランザクションの方向をサブスクライバ起動に制限することが賢明です。サブスクライバがネットサーフィンを開始するとき、HTTP は常にサブスクライバによって起動されるからです。HTTP トランザクションの方向がネットワーク起動である場合、着信 HTTP トラフィックを受信するために、ネットワーク サブスクライバのローカル コンピュータ上の Web サーバが開いていることを意味すると考えられます。HTTP のこのような使用法はネットワーク リソースに負担をかけるので、プロバイダーはこれを禁止することもできます。また、この使用法はネットワークの不正使用であり、サブスクライバとプロバイダーのネットワーク サービス アグリーメントへの違反であるとみなされる場合もあります。

リスト

IP アドレスやホスト名などのネットワーク アドレスを、共通の目的を持ったグループにまとめ、特定のサービスに対応するサブスクライバのネットワーク トランザクションを適用する場合、そのグループのことを *リスト*といいます。システムに複数のリストが存在する場合があります。複数のリストを一括したマスター リストのことを、リスト アレイといいます。

次に、リストの例を示します。

- プロバイダーが望ましくないと判断する有害コンテンツを提供する Web サイトのリスト。「児童向けフィルタリング サービス」を定義して、これらのサイトへのアクセスを禁止することができます。
- ダウンロード転送速度を一定の帯域幅レートに制限したり、使用できる同時セッション数を制限したりすることをプロバイダーが希望する FTP サイトのリスト（ネットワーク アドレスを含む）。

ネットワーク アドレスを含んだリストの指定方法を、次の表に示します。

表 4-2　ネットワーク アドレス タイプの例

ネットワーク アドレス	例
IP アドレス	123.123.3.2
IP アドレス範囲（およびマスク）	マスク付き IP アドレス範囲は、123.3.123.0/24 という形式で指定できます。この場合、IP アドレスの先頭 24 ビットは指定するとおりで、残りの 8 ビット、すなわち 256 個の IP アドレスが、この範囲に含まれることになります。
ホスト名	www.cnn.com

ルール

サービスのコンポーネント（リスト、プロトコル、起動側など）は、ネットワークを通過するトランザクションの解釈方法だけをシステムに指示します。これらは、サービスに適用すべきアクションは指示しません。*ルール*は、ルールの条件が満たされたときに実行すべきアクションを指定する、サービスの条項です。

サービスは時間ベースの情報にも作用する必要があるので、開発者やプログラマが定義する時間枠ごとに 1 つずつ、最高 4 つのサブルールを定義できるようになっています。

前述したように、サービスはサービス エレメントで構成されます。ルールは個々のエレメントに対してではなく、サービス全体に適用されるということを理解しておく必要があります。このことは、開発者やプログラマがサービス コンフィギュレーションをプランニングするときに重要となる、システム解析およびサービス設計上の考慮事項です。

一般にルールとは、特定のサービスのネットワーク トランザクションの取り扱い方法に関する SCE プラットフォームへの指示の集合です。禁止すべきトランザクションを指定するルールや、一定の帯域幅を許容すべきトランザクションを指定するルールがあります。また、合計容量やセッション数の限界を定義するルールもあります。その上で、さらに別の制限をトランザクションに加えることができます。また、課金や解析のためのトランザクション レポートの生成方法を指定するルールもあります。

パッケージ

パッケージは、サブスクライバに提供するサービスのグループを定義します。パッケージには、サービス別のシステム動作の定義が含まれます。具体的には、ネットワーク トランザクションに関する制限、トランザクションの優先順位に関するガイドライン、トランザクション レポートの生成に関する指示などです。このような動作を*ルール*で定義します。

（注） パッケージの処理は、ライセンスによって異なります。

ネットワーク上のサブスクライバごとに、そのサブスクライバが所属しているパッケージの参照が提供されます。ネットワーク トランザクションがいずれかのサービス エレメントの定義を満たす場合、システムはそのトランザクションを特定のサービスに対応付けます。また、システムはトランザクションのネットワーク ID に従って、そのトランザクションに関係するサブスクライバを識別します。システムがトランザクションのネットワーク ID を認識すれば、サブスクライバが所属するパッケージを判別することができ、サブスクライバのネットワーク トランザクションのサービスに適切なルールを適用できます。

次の表に、パッケージの例、対応するサービス、およびサービス パラメータを示します。

表 4-3　パッケージ、サービス、および対応するパラメータの例

パッケージ名	サービス	アクセス	帯域幅レート	容量割り当て	トランザクション単位の同時セッション数
adsl-bronze	閲覧	許可	無制限	200 MB/Wk（メガバイト / 週）	無制限
	Web ホスティング	許可	20 Kb/s（キロビット / 秒）	無制限	3
	ローカル ストリーミング	禁止	N/A	N/A	N/A
	FTP	許可	30 Kb/s	無制限	5
	ビデオ会議	許可	30 Kb/s	600 MB/Wk	5
adsl-gold	閲覧	許可	無制限	無制限	1
	Web ホスティング	許可	80 Kb/s	500 MB/M（メガバイト / 月）	3
	ローカル ストリーミング	許可	無制限	無制限	10
	ビデオ会議	許可	無制限	無制限	30

グローバル コントローラ

SCAS BB ソリューションでは、帯域幅の制御はグローバル制御およびサブスクライバ帯域幅の制御という 2 段階で実行されます。

SCE プラットフォームは、グローバル コントローラという仮想キューを使用して帯域幅を制御します。1 つのインターフェイス（アップストリーム / ダウンストリーム）に最大 16 個のグローバル コントローラを設定できます。グローバル コントローラなので、特定のパッケージに設定がリンクされるのではなく、システム全体が設定の対象になります。

サブスクライバ レベルでの帯域幅の制御とは対照的に、グローバル コントローラの目的は、「Gold サブスクライバ トラフィック全体」や「P2P トラフィック全体」のように、大規模でグローバルなトラフィック容量に制限を加えることです。それぞれのグローバル コントローラは、特定のタイプの全トラフィックに割り当てられる合計システム帯域幅のパーセンテージを表します。具体例としては P2P トラフィックがあります。P2P トラフィックの容量が増えたために、多くの ISP で重大な問題を引き起こしています。グローバル コントローラを使用すれば、システム上の P2P トラフィックの合計量をトラフィック帯域幅全体に対する任意のパーセンテージに制限できます。P2P トラフィックが消費するトラフィック帯域幅の量を、一定に保ちながら制御することが可能になります。

サブスクライバ BW コントローラ

サブスクライバ帯域幅コントローラ（BW コントローラ）は、サブスクライバ全体のトラフィック、またはその一部分を制御します。BW コントローラの設定に使用する主要なパラメータは、次の 2 つです。

- BW コントローラが制御するトラフィックに対して許容すべき最低限の帯域幅
- そのトラフィックが使用できる最大の帯域幅

サブスクライバ BW コントローラは、次の 2 つのレベルで帯域幅を実施します。

- 第 1 レベル、プライマリ BW コントローラ（合計）— *プロバイダー*がサブスクライバに対して実施する帯域幅サービス コンフィギュレーションを指定します。
- 第 2 レベル、BW コントローラ（内部）— *サブスクライバ*がサービスに対して実施を希望するサービス コンフィギュレーションを指定します。

SCAS BB では、各サブスクライバに 1 つずつ独立した BW コントローラのセットが提供されます。そのうち 1 つの BW コントローラは、サブスクライバの合計帯域幅を制御します。この BW コントローラは、第 1 レベルの制御を提供し、プライマリ BW コントローラといいます。

残りの BW コントローラは、そのサブスクライバの一部のサービスの帯域幅を制御します。たとえば、ストリーミング サービスを制御する BW コントローラや、ダウンロードと電子メール サービスをまとめて制御する BW コントローラが考えられます。これらの BW コントローラは、第 2 レベルの制御を提供し、BWC（内部）といいます。

サブスクライバ クォータ バケット

外部クォータ プロビジョニング モードでは、クォータ（割り当て量）アカウンティングはサブスクライバ クォータ バケットを使用して行われます。サブスクライバごとに 16 個のバケットがあり、各バケットを容量またはセッション数で定義できます。サブスクライバが特定のサービスを利用すると、消費した容量またはセッション数が、いずれかのバケットから減らされます。それぞれのサービスでどのバケットを使用するかは、サービス コンフィギュレーションによって決まります。容量バケットの場合、消費量は L3 キロバイトの単位でカウントされます。セッション バケットの場合、消費量はセッション数で表されます。たとえば、閲覧と電子メール サービスはバケット #1 から、P2P サービスはバケット #2 から、それぞれクォータを消費するものとし、それ以外のサービスはどれも特定のバケットに対応付けないように定義することができます。

外部クォータ プロビジョニング システムでは、*Quota Provisioning（QP）API*（「Quota Provisioning API」[p.7-1]）を使用して、各バケットのクォータをダイナミックに変更することができます。たとえば、サブスクライバが追加のクォータを購入したときに、特定のバケットのクォータを増やすことができます。これらのシステムでは、各バケットのクォータ残量を照会することも可能です。たとえば、（ある種の個人 Web ページで）クォータがどれだけ残っているかをサブスクライバに確認させるために、この機能を使用できます。

CHAPTER 5

Service Configuration API

この章では、SCAS Service Configuration API を使用して SCAS BB のサービス設定作業を自動的に行う方法を説明します。この章の目的は次のとおりです。

- SCAS Service Configuration API の主な要素を紹介します。
- SCAS Service Configuration API の使用法、サービス コンフィギュレーションの作成、編集、メンテナンス、管理、および配布機能について説明します。
- SCAS Service Configuration API の使用法を示すコード例を示します。

この章の内容は次のとおりです。

Service Configuration API の概要

*サービス コンフィギュレーション*は、プロバイダーのネットワーク上で Service Control Application Suite for Broadband システムによって実施されるプロバイダーのビジネス ルールを表します。*Service Configuration API* は、*サービス コンフィギュレーション*のプログラミング実装を担うプログラミング エンドです。

サービス コンフィギュレーションを取り扱う作業に Service Configuration API を使用します。SCAS BB Console は、Service Configuration API のプログラミング ツールで作成されています。この API を使用すると、GUI では手作業で行うことになる定常的なサービス設定作業を、自動的に行うことができます。

サービス コンフィギュレーションの定義から SCE プラットフォームでコンフィギュレーションをアクティブ化するまでの、Service Configuration API による主なプログラミング ステップは次のとおりです。

ステップ 1　新しいサービス コンフィギュレーションを定義するか、または SCE から既存のサービス コンフィギュレーションを取得します。

ステップ 2　サービスを定義して、サービス名を割り当てます。

ステップ 3　プロトコルおよびリストを定義します。

ステップ 4　サービス エレメントを定義し、それらのエレメントにプロトコル、方向、およびリスト（オプション）を割り当てます。

ステップ 5　サービス エレメントをサービスに追加します。

ステップ 6　パッケージを定義して、パッケージ名を割り当てます。

ステップ 7　パッケージのサービス ルール（違反前および違反後における帯域幅の制限、サービスのイネーブル化またはディセーブル化など）を定義します。

ステップ 8　サービス コンフィギュレーションを SCE に伝播してアクティブ化します。

SCAS Service Configuration API を使用して行う主な作業は、サービス コンフィギュレーションの作成および処理です。これには、オフラインで行うことができるサービスとルールの定義が含まれます。サービス コンフィギュレーションが完成すると、SCE プラットフォームにそのサービス コンフィギュレーションを伝播できます。

上記の各ステップについて、以下の各項で詳しく説明します。この章を『*Subscriber Management API Reference Manual*』と併せて参照することで、SCAS BB サービス コンフィギュレーションを作成、編集、管理、および配布できるようになります。

SCAS API のベース クラス

アプリケーションの作成および取り扱いに使用する、主な SCAS API クラスを次の表に示します。

表 5-1　主な SCAS API クラス

クラス	目的
EngageAPI	クライアント プログラムがシステムのハードウェアおよびソフトウェアに接続できるようにします。ネットワーク接続機能を実行する、エレメント管理作業の出発点です。
SCAS PolicyAPI	サービス コンフィギュレーションを管理します。

SCAS クライアント / サーバの接続

SCAS Java プログラムなどのクライアント アプリケーションが、SCE プラットフォームに接続できるようにするための SCAS API Java クラスは、`Engage` です。

- SCAS ライブラリの組み込み
- SCE プラットフォームへの接続

SCAS ライブラリの組み込み

SCAS Java API クラスを取り扱うには、Java JDK バージョン 1.3 以降が必要です。

次の JAR ファイルが Java class-path に含まれている必要があります。

- `um_core.jar`
- `xerces.jar`
- `regexp.jar`
- `jdmkrt.jar`
- `log4j.jar`
- `engage.jar`

これらのファイルは、SCAS クライアントのインストレーション プロセスの一環としてワークステーションにインストールされます。このインストレーションは、SCAS BB 製品に付属する SCAS BB *クライアント インストレーション CD-ROM* を使用して行います。

SCAS クライアントを `C:\Program Files\Cisco\SCAS BB x.x.x\` フォルダにインストールした場合、これらの JAR ファイルはすべて `C:\Program Files\Cisco\SCAS BB x.x.x\lib` フォルダにあります。

（注） 上記の JAR ファイルのバージョンは、SCAS BB ソリューションの他のコンポーネントのバージョンと統一されている必要があります。ソリューションの新しいバージョンをインストールまたはアップグレードした場合には、API 開発者向け JAR ファイルも必ずアップグレードしてください。

SCAS Java API を使用するには、事前に SCAS Java パッケージをインポートする必要があります。プログラムの先頭に次のコード行を追加して、インポートします。

```
import com.cisco.apps.scas.*;
import com.cisco.apps.scas.common.*;
import com.cisco.apps.scas.policy.*;
```

SCE プラットフォームへの接続

SCE プラットフォームからサービス コンフィギュレーションを取得したり、SCE プラットフォームにサービス コンフィギュレーションを適用したりするには、プラットフォームに接続する必要があります。この接続を可能にする SCAS クラスが、通信およびネットワーク アクティビティの調整を行います。

このクラスの `login` メソッドは、永続的な接続を提供します。このメソッドでは、ユーザ名、パスワード、および接続する SCE のホスト名が必要です。メソッドのいずれかのパラメータが不正な場合、例外が発生します。サービス コンフィギュレーションの取得または適用が終了した時点で、接続を閉じる必要があります。

（注）　作業の終了時に、必ず `logout` メソッドをコールしてください。SCE への接続をオープンのままにしないでください。また、取得や適用のたびに新しい接続を作成するのではなく、Connection オブジェクトをできるだけ再利用してください。

`login` メソッドは、永続的な `login` 接続を確立した後で行う *Service Configuration* API コールのための基本的なパラメータ値を確立します。このメソッドは、`Connection` タイプのオブジェクトを返します。このオブジェクトは、*Service Configuration* API の大部分で使用される SCE のハンドルです。

次に、SCE との接続を確立する Java コードを示します。

```
String username = "admin";
String password = "pcube";
String se = "212.47.174.32";
Connection connection = null;
try{
   connection = Engage.login(se, user, password, Connection.SE_DEVICE);
}catch (ConnectionFailedException e)
{
// login failed - handle exception
}
```

SCE との接続は、終了しないかぎり開いた状態が継続します。SCE との接続を終了するには、次のように `logout` メソッドを使用します。

```
Engage.logout(connection);
```

SCAS *Service Configuration* API はサービス コンフィギュレーションのさまざまな編集タスクを実行しますが、サービス コンフィギュレーションを編集している間は SCE に接続する必要はありません。つまり、サービス コンフィギュレーションはオフラインで準備することができます。サービス コンフィギュレーションの編集が完了したら、SCE に接続してサービス コンフィギュレーションを伝播できます。

SCAS Service Configuration API によるサービス コンフィギュレーションの管理

前項では、SCAS API を使用して SCE に接続（ログイン）および接続を終了（ログアウト）する方法を説明しました。そのほかに実行できる操作としては、サービス コンフィギュレーションの新規作成、サービス コンフィギュレーションの適用などがあります。適用とは、SCE プラットフォームにサービス コンフィギュレーションを伝播して、ネットワーク上でアクティブ化できるようにすることです。

Java の `PolicyAPI` クラスが、SCAS Service Configuration API のサービスの出発点となります。

以下の各項で、パッケージ、サービス、サービス エレメント、プロトコル、リスト、およびルールに対して実行できるさまざまな操作の方法を説明し、擬似コードの例を示します。サービス コンフィギュレーションの一連のコンポーネントは、最終的にパッケージにアセンブルして SCE に伝播します。ここで説明する内容は次のとおりです。

- サービス コンフィギュレーションの取得および適用
- サービス コンフィギュレーションのインポート、エクスポート、および作成
- リスト
- プロトコル
- サービスおよびサービス エレメント
- パッケージ

サービス コンフィギュレーションの取得および適用

プロバイダーのネットワークで、サービス コンフィギュレーションに変更を加える場合には、現在使用しているサービス コンフィギュレーションを変更するのが一般的です。変更を行うには、SCE からサービス コンフィギュレーションを*取得*して、編集作業を行う必要があります。*サービス コンフィギュレーション*の編集はオフラインで行い、SCE に伝播できる状態にします。*サービス コンフィギュレーション*を伝播することを、サービス コンフィギュレーションの*適用*という場合もあります。これは、サービス コンフィギュレーションの元の内容を上書きして、SCE プラットフォーム上で新しいビジネス ルールをアクティブ化することを意味します。

サービス コンフィギュレーションの取得および SCE への適用を行うための、2 つの主なサービス コンフィギュレーション メソッドである `retrievePolicy` および `applyPolicy` を、次のコード例で示します。

```
// retrieve a service configuration
Policy myPolicy =
      PolicyAPI.retrievePolicy(connection);
// apply a policy
PolicyAPI.applyPolicy (connection, myPolicy, SCAS.APPLY_FLAG_OVERWRITE);
```

`retrievePolicy()` メンバー関数の 2 番めのパラメータ *SCE* には、1 台または複数の Service Control プラットフォームを定義できます。`SCAS.Policy` を取得したあと、サービス コンフィギュレーションの内容の検査、変更、保存など、さまざまな機能を実行できます。

SCAS アプリケーションを開発するとき、いくつかの便利な機能を利用して SCAS Service Configuration API アプリケーションを確認できます。たとえば、Java アプリケーションでサービス コンフィギュレーションを変更する場合、`savePolicy` メソッドを使用して、サービス コンフィギュレーションの内容をファイルに保存できます。その後、SCAS Service Configuration Manager GUI アプリケーションを使用してこのファイルを開き、アプリケーションで変更したサービス コンフィギュレーション エレメントを検証できます。

SCAS サービス コンフィギュレーションを適用するには、applyPolicy メソッドをコールします。

サービス コンフィギュレーションのインポート、エクスポート、および作成

SCE プラットフォームから既存のサービス コンフィギュレーションを取得するだけでなく、テキスト ファイルを通じてサービス コンフィギュレーションを取得することもできます。次に、サービス コンフィギュレーションをインポートする例を示します。

```
String filename = "policy.pqb";
BufferedReader br =
    new BufferedReader(new InputStreamReader(new FileInputStream(filename)));
Policy importedPolicy =
    ImportExportUtils.importPolicy(ImportExportUtils.XML_FORMATTER, br);
```

サービス コンフィギュレーションのエクスポートは、アーカイブの目的で、またはサービス コンフィギュレーションを後で編集する場合に行います。次に、サービス コンフィギュレーションをエクスポートする例を示します。

```
String filename = "policy.pqb";
PrintStream print = new PrintStream(new FileOutputStream(new File(filename)));
ImportExportUtils.exportPolicy(policy, ImportExportUtils.XML_FORMATTER, print);
```

一般的には既存のサービス コンフィギュレーションを修正して新しいサービス コンフィギュレーションを作成しますが、必要に応じてサービス コンフィギュレーションを最初から新規作成することもできます。Policy クラスのコンストラクタで、new 演算子を使用してサービス コンフィギュレーションを作成します。次に、このようなサービス コンフィギュレーションの作成例を示します。

```
Policy myPolicy = new Policy("");
```

リスト

ここでは、リストの使用例を示します。有害な内容の Web サイトのリストを作成し、既存のサービス コンフィギュレーションのリスト アレイに追加します。この例で作成するリスト名は *Offensive Content Website List* であり、リスト要素を追加しています。最後にこの新しいリストを、サービス コンフィギュレーションのリスト アレイに追加します。後の段階で、このリストに含まれるホスト名に関しては *HTTP Browsing Protocol* でサービスをアクセスできないようにするルールを実施できます。

Hostlist クラスのコンストラクタでは、パラメータ リスト name およびオプションの description を指定します。リストに新しい要素を追加するには、HostList クラスの add メソッドを使用します。次のように、getListArray メソッドを使用して listArray を取得し、新しく作成したリスト offensiveList をリスト アレイに追加します。

```
. . .
String name = " Offensive Content Website List ";
String description = " A list of offensive website hosts. ";
HostList offensiveList = new HostList(name, description);
try
{
  offensiveList.add(new HostListItem("www.offensive-content.com"));
  offensiveList.add(new HostListItem("www.more-offensive-content.com"));
} catch(DuplicateItemException e)
{
// handle duplicate exception
}
ListArray listArray = myPolicy.getListArray();
try
{
  listArray.addList(offensiveList);
} catch(DuplicateItemException e)
{
// handle duplicate exception
}
```

上記のコード例で使用されている各要素について、次の各項で説明します。内容は次のとおりです。

- リスト アレイの取得
- リスト アレイのナビゲート
- リストのタイプの判別
- リスト アレイへの要素の追加

リスト アレイの取得

次の行で、リスト アレイを取得しています。

```
ListArray listArray = myPolicy.getListArray();
```

`getListArray` メソッドにより、タイプ `ListArray` のリスト アレイを取得します。リスト アレイを取得したあと、リスト サイズを調べたり、リストの個々の要素を取得したりすることができます。

リスト アレイのナビゲート

次の行で、`listArray` の要素を反復的に取得しています。

```
for (int i = 0; i < listArray.getSize(); i++)
{
      Object o = listArray.getElementAt(i);
}
```

上記のループによって、`listArray` の各要素が返されます。`getSize` メソッドは、`ListArray` のサイズを返します。リストの個々の要素の `getElementAt` メソッドにより、`Object` が割り当てられます。

リストのタイプの判別

`getElementAt` メソッドによって返されるオブジェクトは、`HostList` クラスまたは `IPRangeList` クラスのインスタンスである可能性があります。`IPRangeList` は、IP 範囲のリストです。次に、このメソッドを使用してリストのタイプを判別する例を示します。

```
if (o instanceof HostList)
{
    HostList hostlist = (HostList) o;
    System.out.println("Host List");
} else // IPRangeList
{
    IPRangeList iplist = (IPRangeList) o;
    System.out.println("IP Range List");

}
```

リスト アレイへの要素の追加

次に、リストに新しい要素を追加する例を示します。

```
// To add a new host list item
hostlist.add(new HostListItem("www.cnn.com"));
```

リストのグループの最初のリストを取得したと仮定すると、*www.cnn.com*（http://www.cnn.com）がリストに追加されます。

プロトコル

SCE プラットフォームは、サービスに対応付けられたネットワーク トランザクションのプロトコルに反応します。サービス コンフィギュレーションにはサービスのリストが含まれます。サービスのプロトコルを作成する方法としては、プロトコルを新しく定義する方法と、既存のプロトコルを拡張してポートを追加する方法の 2 通りがあります。また、プロトコルを定義するダイナミック シグニチャ スクリプトをインポートまたは削除することもできます。以下の各項では、次の事項について説明します。

- プロトコルの定義
- プロトコルへのポートの追加
- ダイナミック シグニチャ スクリプト

プロトコルの定義

プロトコルの認識は、一般的なネットワーキング慣例に基づいて行われます。たとえば、サーバがポート 666 でトランザクションを待ち受ける場合、ポート 666 はタイプ DOOM のプロトコルを待ち受けるのが慣例となっています。提供するサービスに関して慣例に従うと、双方にとって便利です。

次に、`quake` という名前のプロトコルを、ポート 26000、トランスポート タイプ TCP で定義する例を示します。

```
// create a new protocol and pass it a reference to a service configuration
Protocol quakeProtocol = new Protocol(myPolicy);
// set the protocol name
try
{
    quakeProtocol.setName("quake");
} catch(DuplicateException de)
{
    //a Protocol with such a name already exists in the service configuration
    // handle exception
} catch(ItemNotFoundException infe)
{
    //item was missing while validating rename: service configuration corrupt
    // handle exception
}
// add new port and transport type
try
{
    quakeProtocol.add(new PortListItem(26000, Consts.TRANSPORT_TYPE_TCP));
} catch(DuplicateItemException e)
{
   // the service configuration has a Protocol with such a defined port
   // handle exception
}
// add the quake protocol to the service configuration 's protocol list
try
{
    protocols.add(quakeProtocol);
} catch(DuplicateItemException e)]
{
    // the service configuration already has this Protocol
    // handle exception
}
```

上記の例では、`myPolicy` という名前のサービス コンフィギュレーションの参照が存在し、`Protocol` クラスのコンストラクタのパラメータとして渡されていることを前提としています。このコンストラクタによって返される `quakeProtocol` の参照を使用して、プロトコルの名前を設定します。

次の行では、プロトコルのポート番号 26000、トランスポート タイプ TCP を指定しています。

```
quakeProtocol.add(new PortListItem(26000, Consts.TRANSPORT_TYPE_TCP));
```

次の行では、このプロトコルをサービス コンフィギュレーションのプロトコル リストに追加しています。

```
protocols.add(quakeProtocol);
```

以上でプロトコルの定義は終了です。次項では、既存のプロトコルにポートを追加する方法を説明します。

プロトコルへのポートの追加

使用されていない*ポート*を、既存のプロトコルに*追加*できます。基本的にそのポートはそのプロトコルに対応付けられ、プロトコルの範囲が拡張されます。

次に、サービス コンフィギュレーションのプロトコル リストを取得し、HTTP Browsing という名前のシステム定義プロトコルを検索する例を示します。このプロトコルを見つけたあと、ポートを追加してプロトコル定義を拡張します。

```
    . . .
// get protocol list
ProtocolArray protocols = myPolicy.getProtocolList();
// get a protocol called "HTTP Browsing":
try
{
    Protocol http = protocols.getProtocol ("HTTP Browsing");
} catch(ItemNotFoundException infe)
{
    // the service configuration does not have a Protocol with such a name
    // handle exception
}
// add port 8082 to HTTP Browsing
try
{
    http.add(new PortListItem (8082, Consts.TRANSPORT_TYPE_TCP));
} catch(DuplicateItemException e)
{
   // the service configuration has a Protocol with such a defined port
   // handle exception
}
    . . .
```

この例では、HTTP Browsing という名前のプロトコルが存在することを前提としています。次の行で、Protocol クラスの getProtocol メソッドにより、プロトコル アレイを表すハンドルが返されます。

```
ProtocolArray protocols = myPolicy.getProtocolList();
```

次の行で、特定のプロトコルを名前によって検索します。

```
Protocol http = protocols.getProtocol ("HTTP Browsing");
```

クラス Protocol の add メソッドでは、ポート番号およびトランスポート タイプを指定する必要があります。上記の例では、ポート 8082 をトランスポート タイプ TCP で、既存のサービス HTTP Browsing に追加しています。次の行で、プロトコルにポートを追加します。

```
http.add(new PortListItem (8082, Consts.TRANSPORT_TYPE_TCP));
```

サービス HTTP Browsing が拡張されました。ポート 8082 で配信される HTTP トランザクションをサービス ルールによって処理するには、この拡張が必要です。

サービスおよびサービス エレメント

次に、2 つのサービス エレメントのあるサービスを作成する例を示します。この例は、1 つの *サービス*に複数の *サービス エレメント*の対応付けが可能であることを示しています。

作成するサービス エレメントの 1 つは Doom、もう 1 つは Quake というゲームです。どちらも有名なゲームで、これらを Gaming Service というサービスに追加します。

```
. . .
Service gamingService = new Service(myPolicy); // create a service
try
{
    gamingService.setName("Gaming Service"); // set service name
} catch(DuplicateException de)
{
    // a Service with such a name already exists in the service configuration
    // handle exception
} catch(ItemNotFoundException infe)
{
    //item was missing while validating rename: service configuration corrupt
    // handle exception
}
// create a service element for Subscriber-Initiated Doom
try
{
    gamingService.addProtocol("doom",
                              ServiceElement.DIRECTION_SUBSCRIBER_INITIATED);
} catch(ItemNotFoundException e)
{
    // there is no such Protocol in the service configuration
    // handle exception
} catch(DuplicateItemException e)
{
    // there is already a Service with such a Protocol and direction in the service
configuration
    // handle exception
}
// create a service element for Subscriber-Initiated Quake
try
{
    gamingService.addProtocol("quake",
                              ServiceElement.DIRECTION_SUBSCRIBER_INITIATED);
} catch(ItemNotFoundException e)
{
    // there is no such Protocol in the service configuration
    // handle exception
} catch(DuplicateItemException e)
{
    //There is already a Service with such a Protocol and direction in the service
configuration
    // handle exception
}
// add service to service configuration
try
{
   myPolicy.getServiceList().add(gamingService);
} catch(DuplicateItemException e)
{
    //There is already a Service in the service configuration
    // handle exception
}

. . .
```

上記のコード例で使用されている各要素について、次の各項で説明します。

- サービスの作成
- サービス エレメントの定義
- サービス コンフィギュレーションへのサービスの追加

サービスの作成

次の行で、Service を含むクラスをインスタンシエートしています。

```
Service gamingService = new Service (myPolicy); // create a service
```

myPolicy という名前の Policy 参照が、Service コンストラクタのパラメータとして渡されていることを前提としています。

次の行で、サービス名を定義しています。

```
gamingService.setName("Gaming Service"); // set service name
```

サービス エレメントの定義

次の行で、新しいサービス エレメントを作成しています。

```
gamingService.addProtocol("doom",
                          ServiceElement.DIRECTION_SUBSCRIBER_INITIATED);
```

このサービス エレメントは、サブスクライバ起動による doom プロトコルのすべてのネットワーク トランザクションに適用されます。ただし、特定のネットワーク アドレスに対して実行されたネットワーク トランザクションに限り、このサービス エレメントを適用するように指定することができます。それには、次のようにサーバ アドレスのリストをサービス エレメント付きで指定します。

```
int listIndex =
    myPolicy.getListArray().getListIndex("gaming servers list");
if(listIndex == -1)
{
    // there is no such list
    // handle error
}
int serviceElementIndex =
gamingService.getServiceElementArray.getIndexOfItem("doom", ServiceElement.
DIRECTION_SUBSCRIBER_INITIATED);
if(serviceElementIndex == -1)
{
    // There is no such ServiceElement
    // handle error
}
try {
    gamingService.addList(serviceElementIndex,listIndex);
} catch(ItemNotFoundException e)
{
// handle error
} catch(DuplicateItemException e)
{
// handle error
}
```

サービス コンフィギュレーションへのサービスの追加

最後のステップとして、サービス コンフィギュレーションにサービスを追加します。次の行で、サービス コンフィギュレーションに gamingService を追加しています。

```
// add service to service configuration
myPolicy.getServiceList().add(gamingService);
```

パッケージ

（注）　パッケージの定義は、ライセンスに依存します。ライセンスの容量制限により、そのライセンスで規定されているパッケージ数より多く定義することはできません。

ルールおよびパッケージの定義を始める前に、前述の手順に従って、サービスを定義し、サービス コンフィギュレーションにサービスを追加する必要があります。

次に、パッケージおよびルールを定義し、これらを使用するためのコード例を示します。1 つのサービス コンフィギュレーションで、Family Package および Bachelor Package という 2 つのパッケージを作成しています。

この例では、Parental Watch List を含む Parental Watch Service を作成済みであることを前提としています。Family Package では、Parental Watch Service をイネーブルにします。このパッケージに加入するサブスクライバは、Parental Watch List で定義されている検閲済みのコンテンツにアクセスできません。Bachelor Package では、Parental Watch Service をイネーブルにしないので、これらのサイトへのアクセスが認められます。

```
/* Define package 1 */
Package family = new Package(myPolicy); // create a package
try
{
    family.setName("Family Package"); // set package name
} catch (DuplicateItemException e)
{
    // there already is a Package in service configuration with such a name
    // handle exception
}
ServiceRule serviceRule = new ServiceRule(policy);
// adding service to rule
try
{
    serviceRule.setServiceName("Parental Watch Service");
} catch(ItemNotFoundException e)
{
    // no such service
    // handle exception
}
// get the default rule - it will apply to all the time-based rules
// since no time-based rules are created
Rule rule = serviceRule.getDefaultRule();
// set rule state to be enable
rule.setState(Rule.RULE_STATE_ENABLED);
// set rule action to be block
rule.setPreBreachAccessMode(Rule.ACCESS_BLOCK);
// get package's service rule array
ServiceRuleArray serviceRuleArray = family.getServiceRuleList();
// add service rule to the package's serviceRuleArray
try
{
    serviceRuleArray.add(serviceRule);
} catch(DuplicateItemException e)
{
    // there already is such a rule
    // handle exception
}
/* Define package 2 */
Package bachelor = new Package(myPolicy); // create a package
try
```

```
{
    bachelor.setName("Bachelor Package"); // set package name
} catch (DuplicateItemException e)
{
    // there already is a Package in service configuration with such a name
    // handle exception
}
serviceRule = new ServiceRule(policy);
// adding service to rule
try
{
    serviceRule.setServiceName("Parental Watch Service");
} catch(ItemNotFoundException e)
{
    // no such service
    // handle exception
}
// get the default rule - it will apply to all the time-based rules
// since no time-based rules are created
rule = serviceRule.getDefaultRule();
// set rule state to be enable
rule.setState(Rule.RULE_STATE_ENABLED);
// set rule action to be block
rule.setPreBreachAccessMode(Rule.ACCESS_ADMIT);
// get package service rule array
serviceRuleArray = bachelor.getServiceRuleList();
// add service rule to the package's serviceRuleArray
try
{
    serviceRuleArray.add(serviceRule);
} catch(DuplicateItemException e)
{
    // there already is such a rule
    // handle exception
}
// add the packages to the service configuration
try
{
    policy.getPackageList().add(family);
    policy.getPackageList().add(bachelor);
}catch(DuplicateItemException e)
{
    //there are already such packages
    //handle exception
}
```

上記のコード例で使用されている各要素について、次の各項で説明します。内容は次のとおりです。

- パッケージの作成およびパッケージ名の設定
- サービス ルールの定義
- 違反
- 例 — FTP サービス ルール
- アグリゲーション
- 帯域幅コントローラ
- 違反レポート
- デフォルトの時間枠と非デフォルトの時間枠
- ブロックおよびリダイレクト

パッケージの作成およびパッケージ名の設定

パッケージを作成するには、new 演算子を使用します。次の行で、Package を含むクラスをインスタンシエートしています。

```
Package family = new Package(myPolicy); // create a package
```

新しいパッケージを作成したあと、次の行でパッケージに名前を付けています。

```
family.setName("Family Package"); // set package name
```

サービス ルールの定義

次の行で、レジデンシャルな getRule メソッドのパラメータで指定したルールの参照が返されます。

```
ServiceRule pw =residential.getRule("Parental Watch Service");
```

違反

ネットワーク トランザクションに指定されている*制限*を超過すると、*違反*が発生します。サービスのルールを宣言するときに、違反を定義します。帯域幅の容量制限またはキロビット / 秒（Kb/s）転送速度の制限を超過した場合や、サービスであらかじめ定義されているその他の制限を超過した場合を、違反とみなすことができます。違反を定義する目的は、このような制限に違反した場合に実行すべきアクションを指定することです。

ルールには、*違反前（pre-breach）*および*違反後（post-breach）*の 2 つのモードがあります。サービス ルールの制限に達しないときは *pre-breach アクション*が実行され、制限に達して超過したときには *post-breach アクション*が実行されます。

これらのアクションの例としては、帯域幅容量の制限、サービスの拒否、RDR レポートのトリガー、および Web サイト リダイレクションがあります。すべての pre-breach ファンクション コール（メソッド）に、対応する post-breach ファンクション コールがあります。

違反はデフォルトの状態では非アクティブなので、違反をアクティブにしてから適用する必要があります。次の行で、setPreBreachAccessMode メソッドのフラグを true に設定することにより、違反前の条件をアクティブにしています。

```
rule.setPreBreachAccessMode(Rule.ACCESS_ADMIT);
```

例 — FTP サービス ルール

次に、FTP サービス用のルールを設定する例を示します。この例では、FTP File Downloading という名前のサービスが存在することを前提としています。これらのルールにより、帯域幅は 5 Kb/s、1 日あたり 100 MB に制限されます。どちらかの制限に達すると、サービスを禁止し、RDR をトリガーします。この例では、RDR は 1 日あたりの合計をレポートで使用すると指定しています。最後にサービス コンフィギュレーションを適用し、そのあとで接続を終了します。

```
// create new package
Package residential = new Package(myPolicy);
// add the packages to the service configuration
try
{
policy.getPackageList().add(residential);
} catch(DuplicateItemException e)
{
// there already are such packages
// handle exception
}
// set service name
try
{
    residential.setName("residential");
} catch (DuplicateItemException e)
{
    // there already is a Package with such a name in service configuration
    // handle exception
}
ServiceRule serviceRule = new ServiceRule(policy);
// adding service to rule
try
{
    serviceRule.setServiceName("Parental Watch Service");
} catch(ItemNotFoundException e)
{
    // no such service
    // handle exception
}
// get the default rule - it will apply to all the time-based rules
// since no time-based rules are created
Rule rule = serviceRule.getDefaultRule();
// set rule state to be enable
rule.setState(Rule.RULE_STATE_ENABLED);
// set rule action to be admitted
rule.setPreBreachAccessMode(Rule.ACCESS_ADMIT);
// limit daily volume to approximately 100 MB
rule.setBreachVolumeLimit(100000);
// set rule post-breach action to be blocked
rule.setPostBreachAccessMode(Rule.ACCESS_BLOCK);
// generate RDR when limit is breached
rule.setBreachReportEnabled(true);
// get package service rule array
ServiceRuleArray serviceRuleArray = residential.getServiceRuleList();
// add service rule to the package's serviceRuleArray
try
{
    serviceRuleArray.add(serviceRule);
} catch(DuplicateItemException e)
{
    // there is already such a rule
    // handle exception
}
// tell the package to use daily aggregation periods
residential.setAggregationPeriod(Package.AGGREGATED_PERIOD_DAILY);
// apply changes to the SCE Platform
PolicyAPI.applyPolicy(connection, myPolicy, SCAS.APPLY_FLAG_OVERWRITE);
// close the connection
SCAS.logout(con);
```

上記の例でわかるように、総容量はサービス トランザクションで使用されたキロバイト数または使用されたネットワーク セッション数のどちらでも測定できます。どちらかの制限に達した時点で、サブスクライバの使用帯域幅レートを変更したり、レポート（RDR）を送信したり、サブスクライバをサービスから除外したりできます。

以下の各項で、このプログラムの各部分について個々の要素を説明します。

アグリゲーション

*アグリゲーション*期間は、パッケージごとにグローバルで定義します。アグリゲーションは、RDR が作成されたときレポート生成に使用されます。アグリゲーションは、月、週、日、または時間ごとに測定できます。

同じサービスで、パッケージごとに異なるアグリゲーション期間を使用できます。つまり、1 つのパッケージで FTP Download Service の 1 日あたりのトータルを指定し、同じ FTP Download Service を使用するもう 1 つのパッケージでは 1 時間あたりのトータルを指定することが可能です。このアグリゲーション（トータルでの使用状況に関する情報）は、サブスクライバ宛の項目別の請求書または取引明細書に利用できます。

次の行で、1 日あたりのトータルを使用して RDR を生成することを指定しています。

```
residential.setAggregationPeriod (Package.AGGREGATED_PERIOD_DAILY);
```

クォータの配分はパッケージごとに定義します。したがって、パッケージが 2 つのサービス（たとえば FTP Download Service および Video Conferencing Service）で構成される場合、アグリゲーションには両方のサービスの合計を反映させます。これはパッケージを設計する際に考慮すべき事項の 1 つです。

上記の例では、サービス コンフィギュレーションをオフラインで編集しているので、一連のコマンドは順不同です。したがって、変更後のサービス コンフィギュレーションを SCE の SCE プラットフォームに伝播するまで、コマンドは実行されません。

帯域幅コントローラ

帯域幅コントローラ（別名メーター）は、特定のサービス トランザクションが使用している容量（Kbps）を測定するためのフロー ルール ベースの機能です。1 フローのトランザクションでも、いくつかのフローを同時に使用するトランザクションでも測定可能です。帯域幅コントローラのルールは、フロー ベース ルール、時間ベース ルール、または両方の組み合わせにすることができます。

帯域幅コントローラには、方向性があります。帯域幅コントローラはそれぞれ、アップストリーム方向またはダウンストリーム方向のどちらかで容量を制御します。この方向は API で設定できます。

フロー数（同時セッション数）およびそれらの合計の使用帯域幅レートがわかれば、プロバイダーがシステム サービスをより強力に制御して、ネットワーク リソースの利用方法に関してより影響を及ぼすことができるので便利です。たとえば、帯域幅コントローラの機能によって、サブスクライバが特定の帯域幅レートを超過しないかぎり、使用できるフロー数を無制限にするという指定が可能です。

ネットワーク トランザクションで使用されるフロー数を制御すれば、たとえば RTPS プロトコルを使用するインターネット ビデオ アプリケーションのデジタル ブロードキャストなどに役立ちます。RTSP は、ビデオ トラフィックのいくつかの同時フローを維持できます。帯域幅コントローラは、サブスクライバが特定のレート制限を超えないように指定するためのツールをプロバイダーに提供します。

次に、フロー レベルでの帯域幅コントローラ機能の使用例を示します。

```
// assigning BWController number 3 to the Rule in upstream direction
ftpRule.getDefaultRule().setPreBreachUpstreamBWControllerIndex(3);
```

違反レポート

*違反*は、システムで指定された制限を超過した場合に発生します。違反が検出された場合に実行できるアクションは、2 通りあります。それは、(a) サービスを禁止（ブロック）する、または (b) サブスクライバのトランザクションまたはサービスの配信方法に何らかの変更を加えることです。

したがって、システムには 2 タイプのレポート生成機能があります。それは、(a) ブロック レポート、または (b) 違反レポートです。サービスで違反が発生するたびに、違反のタイプに応じてどちらかのレポートをトリガーするように、システムを設定できます。

違反が発生したときに、関数のパラメータを true に設定することによって RDR の作成をトリガーする例を次に示します。

```
Rule.setBreachReportEnabled(true);
```

違反が発生するたびに RDR を生成するのではなく、サブスクライバに電子メールで通知するように設定することもできます。

デフォルトの時間枠と非デフォルトの時間枠

ここでは、時間枠、デフォルトのルール、非デフォルトのルール、およびデフォルトのルール関数を使用する場合について説明します。

時間枠を使用すると、時間によって異なった動作を実行するルールを設定できます。時間枠の例としては、夕方の時間枠、夜間の時間枠、および週末の時間枠があります。SCAS システムでは、最大 4 つの時間枠を定義できます。

API を使用して、時間枠の例外に基づいて情報を処理するアプリケーションを作成できます。たとえば、通常の曜日に基づくスケジュールでは週末料金が金曜日の午後 4 時 54 分から始まるのに対し、31 日水曜日の午後 3 時から翌日の午後 6 時まで、無制限の帯域幅を使用できる週末料金のビデオ会議サービスのために時間枠を変更するアプリケーションなどです。

`getDefaultRule` メソッドを使用すると、そのルールはすべての時間枠に対してグローバルに適用されます。次に、`getDefaultRule` メソッドを使用してグローバルな時間枠ルールを設定する例を示します。

```
// set rule post-breach action to block access
rule.setPostBreachAccessMode(Rule.ACCESS_BLOCK);
```

非デフォルト ルールの例は次のとおりです。サブスクライバがサービスへのアクセスを拒否されるようにデフォルトのルールが定義されていると仮定します。さらに、2 つの時間枠を作成したと仮定します。1 つは平日の 2200 ～ 0600 であり、もう 1 つは金曜日の同じ時間帯です。平日の時間枠ではサービスへのアクセスを許可し、金曜日にはサービスへのアクセスを禁止するように、例外を作成できます。

次のコード例では、T1 という 1 つの時間枠を作成しています。デフォルトのルールはすべての時間枠に適用されるので、時間枠 T1 を使用してその例外を設定し、定義した期間中はサービスへのアクセスを許可しています。

```
    . . .
// creates a new Package
Package myPackage = new Package(myPolicy);
// add the packages to the service configuration
try
{
    policy.getPackageList().add(myPackage);
} catch(DuplicateItemException e)
{
    // there already are such packages
    // handle exception
}
// creates new service rule
ServiceRule serviceRule = new ServiceRule(policy);
// adds service to rule
serviceRule.setServiceName(serviceName);
// gets the default rule - it will apply to all time based rules
// since no time based rules are created
Rule defaultRule = serviceRule.getDefaultRule();
// set default rule state to be disable
defaultRule.setState(Rule.RULE_STATE_DISABLED);
// announce the need for configuring specific behavior to T1
Rule t1Rule = null;
Try
{
  t1Rule = serviceRule.addTimeFrameRule(TimeFrame.T1);
} catch(ItemNotFoundException e)
{
// handle exception
} catch(DuplicateItemException e)
{
// handle exception
}
// set T1 rule state enabled
t1Rule.setState(Rule.RULE_STATE_ENABLED);
// adds service rule to the package's serviceRuleArray
myPackage.setServiceRule(serviceRule);
    . . .
```

ブロックおよびリダイレクト

block-and-redirect 関数は、HTTP、RTSP など、リダイレクトが可能なプロトコルを使用するサービスにのみ関係します。この関数を使用すると、サブスクライバがサービス パッケージに含まれていないサービスへのアクセスを試みた場合に、サブスクライバのトランザクションを別のネットワーク アドレスにリダイレクトできます。そのネットワーク アドレスまたは Web サイトで、サービスへのアクセス方法についての有益な情報を提供できます。

```
// assign the default redirect string of the "HTTP Browsing" protocol
// to be redirected to a certain URL
try
{
    policy.setProtocolRedirectString("HTTP Browsing","http://www.mySite.com/
redirect.html");
} catch (ItemNotFoundException e)
{
    //handle exception
} catch (MalformedURLException e)
{
    //handle exception
}
// creates new service rule
ServiceRule serviceRule = new ServiceRule(policy);
// adds service to rule
serviceRule.setServiceName("http browsing service");
// gets the default rule - it will apply to all time-based rules
// since no time-based rules are created
Rule defaultRule = serviceRule.getDefaultRule();
// set default rule state to be enable
defaultRule.setState(Rule.RULE_STATE_ENABLED);
// set rule post-breach action to block and redirect
defaultRule.setPostBreachAccessMode(Rule.ACCESS_BLOCK_AND_REDIRECT);
```

例 — サービスの追加とサービス コンフィギュレーションの適用

次の例は、ネットワーク プロバイダーがサブスクライバの内部ネットワーク内でホスティングされるゲーム アプリケーション サーバ上でサービスを設定する方法を示しています。この例では、テンプレートとして Doom ゲーム アプリケーションを使用しています。この Java アプリケーションの機能は次のとおりです。

ステップ 1　新しいサービス コンフィギュレーションを作成します。

ステップ 2　*Local Doom* というサービス エレメントを定義します。

ステップ 3　Doom サービスをサブスクライバ起動にします。

ステップ 4　この Java アプリケーションは、サービスをホスティングするサブスクライバ ネットワーク内のローカル サーバの IP アドレスのリストをマッピングします。

ステップ 5　Java アプリケーションは、サブスクライバによるサービスへの接続時間に基づいて、課金レコードをトリガーします。

```
import com.cisco.apps.scas.Connection;
import com.cisco.apps.scas.ConnectionFailedException;
import com.cisco.apps.scas.SCAS;
import com.cisco.apps.scas.PolicyAPI;
import com.cisco.apps.scas.common.DuplicateItemException;
import com.cisco.apps.scas.common.ItemNotFoundException;
import com.cisco.apps.scas.policy.IPListItem;
import com.cisco.apps.scas.policy.IPRangeList;
import com.cisco.apps.scas.policy.ListArray;
import com.cisco.apps.scas.policy.Package;
import com.cisco.apps.scas.policy.PackageArray;
import com.cisco.apps.scas.policy.Policy;
import com.cisco.apps.scas.policy.PortListItem;
import com.cisco.apps.scas.policy.Protocol;
import com.cisco.apps.scas.policy.ProtocolArray;
import com.cisco.apps.scas.policy.Rule;
import com.cisco.apps.scas.policy.Service;
import com.cisco.apps.scas.policy.ServiceArray;
import com.cisco.apps.scas.policy.ServiceElement;
import com.cisco.apps.scas.policy.ServiceRule;
import com.cisco.apps.scas.policy.ServiceRuleArray;

/**
 * SCAS API Example
 */
public class Example {

     public static void main(String[] args) {
           Policy policy = new Policy("");

           // get policy package list
           PackageArray packageArray = policy.getPackageList();

           byte ip_mask = 32;

           // create host list item
           IPListItem item1 = new IPListItem("1.1.1.1", ip_mask);
           IPListItem item2 = new IPListItem("2.2.2.2", ip_mask);
           IPListItem item3 = new IPListItem("3.3.3.3", ip_mask);

           // create ip list
           String listName = "doom ips";
           IPRangeList ipList =
                 new IPRangeList(listName, "A list of doom ips", false);

           // add list item to list
           try {
                 ipList.add(item1);
                 ipList.add(item2);
                 ipList.add(item3);
           } catch (DuplicateItemException e) {
                 System.out.println(
                       "Add ip list item to ip list failed :" +
e.getMessage());
                 System.exit(1);
           }

           // get the policy list array and add to it the ip list
           ListArray policyListArray = policy.getListArray();
           try {
                  policyListArray.addList(ipList);
           } catch (Exception e) {
```

```
                  System.out.println(
                        "Add ip list to policy lists failed :" +
e.getMessage());
                  System.exit(1);
      }

      // get Policy Service list
      ServiceArray policyServiceList = policy.getServiceList();

      // create doom protocol
      String protocolName = "doom";
      ProtocolArray policyProtocols = policy.getProtocolList();
      Protocol protocol = new Protocol(policy);

      // set port
      PortListItem doomPort =
              new PortListItem(666, PortListItem.TRANSPORT_TYPE_BOTH);

      try {
              protocol.add(doomPort);
      } catch (DuplicateItemException e) {
              System.out.println(
                      "ERROR - adding port 666 failed :" + e.getMessage());
              System.exit(1);
      }

      try {
              policyProtocols.add(protocol);
      } catch (DuplicateItemException e) {
              System.out.println(
                      "ERROR - adding protocol \""
                              + protocolName
                              + "\" failed:"
                              + e.getMessage());
              System.exit(1);
      }

      // to allow doom to support IP addresses,
      // the generic TCP and UDP must support it
      try {
              policyProtocols.getProtocol(
                      Protocol.GENERIC_UDP_PROTOCOL).setListElementsType(
                      Protocol.LIST_SUPPORT_IP_RANGE_LIST);
      } catch (Exception e) {
              System.out.println(
                      "ERROR - UDP setListElementsType threw exception"
                              + e.getMessage());
              System.exit(1);
      }

      // create new Service
      Service service = new Service(policy);

      String serviceName = "Local Doom";

      try {
              // set the service name, which is its identifier
              service.setName(serviceName);
      } catch (ItemNotFoundException e) {
              System.out.println(
                      "ERROR - service set name threw
ItemNotFoundException :"
```

```
                                + e.getMessage());
                System.exit(1);
        } catch (DuplicateItemException e) {
                System.out.println(
                        "ERROR - service set name threw
DuplicateItemException :"
                                + e.getMessage());
                System.exit(1);
        }

        // add the doom protocol to the service
        try {
                service.addProtocol(protocolName, ServiceElement.DIRECTION_BOTH);
        } catch (ItemNotFoundException e) {
                System.out.println(
                        "ERROR - adding protocol threw
ItemNotFoundException :"
                                + e.getMessage());
                System.exit(1);
        } catch (DuplicateItemException e) {
                System.out.println(
                        "ERROR - adding protocol threw
DuplicateItemException :"
                                + e.getMessage());
                System.exit(1);
        }

        // add the list to the service: the first index is
        // the service element index ñ that is, to which protocol -
        // and the second index is the list index in the policyListArray
        try {
                service.addList(0, ipList.getName());
        } catch (ItemNotFoundException e) {
                System.out.println(
                        "ERROR - adding list threw ItemNotFoundException :"
                                + e.getMessage());
                System.exit(1);
        } catch (DuplicateItemException e) {
                System.out.println(
                        "ERROR - adding list threw DuplicateItemException :"
                                + e.getMessage());
                System.exit(1);
        }

        // add service to policyServiceList
        try {
                policyServiceList.add(service);
        } catch (DuplicateItemException e) {
                System.out.println(
                        "ERROR - adding service to policy service list "
                                + "threw DuplicateItemException :"
                                + e.getMessage());
                System.exit(1);
        }

        // creating a new Package
        Package pack = new Package(policy);

        String packageName = "Game users";
        try {
                pack.setName(packageName);
```

```
      } catch (DuplicateItemException e) {
              System.out.println(
                      "ERROR - package.setName threw
DuplicateItemException");
              System.exit(1);
      }

      // add package to service configuration package array
      try {
              packageArray.add(pack);
      } catch (DuplicateItemException e) {
              System.out.println(
                      "ERROR - add package failed : " + e.getMessage());
              System.exit(1);
      }

      // create new service rule
      ServiceRule serviceRule = new ServiceRule(policy);
      // add service to rule
      try {
              serviceRule.setServiceName(serviceName);
      } catch (ItemNotFoundException e) {
              System.out.println(
                      "ERROR - service rule set service name failed:"
                              + e.getMessage());
              System.exit(1);
      }

      // get the default rule - it will apply to all the time-based rules
      // since no time-based rules are created
      Rule rule = serviceRule.getDefaultRule();

      // set rule state to enabled
      rule.setState(Rule.RULE_STATE_ENABLED);

      // set rule action to be blocked
      rule.setBillingReportEnabled(true);

      // set other Rule parameters
      // . . .

      // get package service rule array
      ServiceRuleArray serviceRuleArray = pack.getServiceRuleList();

      // add service rule to the package's serviceRuleArray
      try {
              serviceRuleArray.add(serviceRule);
      } catch (DuplicateItemException e) {
              System.out.println(
                      "ERROR - adding service rule to the package "
                              + "threw DuplicateItemException : "
                              + e.getMessage());
              System.exit(1);
      } catch (ItemNotFoundException e) {
              System.out.println(
                      "ERROR - adding service rule to the package "
                              + "threw ItemNotFoundException : "
                              + e.getMessage());
              System.exit(1);
      }
```

■ 例 — サービスの追加とサービス コンフィギュレーションの適用

```
               // connect to the SCE Box
               String username = "admin";
               String password = "pcube";
               String se_address = "212.47.174.32";
               Connection connection = null;
               try {
                      connection =
                             SCAS.login(
                                    se_address,
                                    username,
                                    password,
                                    Connection.SE_DEVICE);
               } catch (ConnectionFailedException e) {
                      // login failed - handle exception
               }

               // to apply the service configuration, the parameters are:
               // a connection, the new policy to apply, the SCE to apply to,
               // and a flag stating that the a connection,should be applied
               // (although you might override other applied service
configurations)
               try {
                      PolicyAPI.applyPolicy(connection, policy);
               } catch (Exception e) {
                      System.out.println("ERROR - first apply failed :" +
     e.getMessage());
                      System.exit(1);
               } finally {
                      SCAS.logout(connection);
       }
       System.exit(0);
     }

     private Example() {}
}
```

CHAPTER 6

サブスクライバ統合

(注) この章で説明する概念については、『*SCE 1000/SCE 2000 User Guides*』および『*SM User Guide*』でさらに詳しく説明されています。詳細はこれらのマニュアルを参照してください。

この章では、SCAS BB アプリケーションにおけるサブスクライバ統合について説明します。使用可能なサブスクライバ モードを紹介し、各モードを使用する状況を示します。

この章の内容は次のとおりです。

サブスクライバ モード

SCAS BB システムは、次のいずれかのサブスクライバ モードで運用できます。

- サブスクライバレス
- アノニマス サブスクライバ
- サブスクライバアウェア：次の 2 種類のモードがあります。
  - スタティック
  - ダイナミック

（注） 使用するモードを制御するためのグローバルな設定は存在しないので、各サブスクライバ モードは暗黙的です。希望するモードでシステムを運用する方法については、システム マニュアルを参照してください。

ここでは、各サブスクライバ モードの機能、利点、および使用する状況を（次の表に）要約したあと、それぞれのモードについてさらに詳しく説明します。

表 6-1　サブスクライバ モードの要約

モード	サポートする機能	主な利点	使用する状況
サブスクライバレス	グローバル（デバイス レベル）での解析および制御	すぐに使用できる：サブスクライバの設定が不要 サブスクライバ（または着信 IP アドレス）数に影響されない	グローバルな制御ソリューションまたはサブスクライバ レベルでの解析。例: • ピアリング ポイントでの P2P アップロードを制御する • P2P の合計量を特定のパーセンテージに制限する
アノニマス サブスクライバ	グローバルでの解析および制御 個々のIPアドレス レベルでの解析および制御	すぐに使用できる：サブスクライバが使用する可能性のある IP 範囲の定義のみ必要 統合を必要としないサブスクライバ レベルでの制御	サブスクライバ単位での差別化のない IP レベルの解析または制御を行う場合、およびオフラインでの IP アドレス / サブスクライバ バインディングで十分な場合。例： • サブスクライバ 1 人あたりの P2P を 64 Kbps（キロバイト / 秒）に制限する • トップ サブスクライバを識別する場合（トップ IP アドレスを識別し、RADIUS/DHCP ログと手動 / オフラインで照合する）
サブスクライバアウェア	すべてのシステム機能	サブスクライバ単位での差別化されたダイナミックな制御 使用されている IP アドレスとは無関係な、サブスクライバ レベルでの解析 スタティックに定義した IP 範囲にトラフィックをグループ化して制御 / 解析	次のような場合： • サブスクライバ レベルでトラフィックを制御および解析する • IP アドレスとは無関係にサブスクライバの使用状況をモニタする • サブスクライバ別に異なるサービス コンフィギュレーションまたはパッケージを割り当て、ダイナミックにパッケージを変更する

サブスクライバレス モード

サブスクライバレス モードは、グローバルなデバイス レベルでの制御機能およびリンク レベル解析機能を提供します。サブスクライバレス モードでは、統合は不要であり、モニタ対象のリンクを利用するサブスクライバの総数は、SCE プラットフォームの視点では無制限です。

サブスクライバがまったく存在しない場合にも、SCAS BB が役立ちます。ユーザは*トラフィック ディスカバリ*を使用してネットワーク アクティビティを表示し、グローバル BW リミッタおよび不明のサブスクライバ用のパッケージを使用してキャパシティ制御を実行できます。

アノニマス サブスクライバ モード

アノニマス サブスクライバ モードは、サブスクライバの着信 IP アドレスでネットワーク トラフィックを解析および制御するための手段になります（たとえば、ネットワーク アクティビティを解析してトップ P2P-IP アドレスを突き止めたり、各サブスクライバの P2P トラフィックを 64 Kbps に制限したりできます）。アノニマス サブスクライバ モードでは、統合も、使用する IP アドレスのスタティックな設定も不要です。このモードでは、SCE プラットフォームに IP アドレスの範囲を設定します。システムはこの範囲で IP アドレスをサブスクライバ名として使用し、アノニマス サブスクライバをダイナミックに作成します。

サブスクライバ単位での差別化された制御やサブスクライバ レベルのクォータ追跡が不要な場合、および IP レベルでの解析で十分な場合に、アノニマス サブスクライバ モードを使用します。

（注） 同時にアクティブなアノニマス　サブスクライバの総数は、同時にアクティブなサブスクライバの総数と同じなので、同様のライセンスに準拠します。

アノニマス サブスクライバ モードでは、各サブスクライバに異なるプールから IP アドレスを割り当てることにより、サブスクライバ別に異なるサービス コンフィギュレーションの割り当てがサポートされます。これには、SCE プラットフォームのサービス コンフィギュレーション テンプレート機能を使用します。この機能により、割り当てられた IP アドレスの範囲に応じて、それぞれのアノニマス サブスクライバに異なるパッケージを割り当てることができます。

（注） アノニマス サブスクライバ モードでは、IP アドレスのリサイクル時間（サブスクライバがネットワークからログオフしたあと、そのサブスクライバの IP アドレスが再割り当てされるまでの時間）が十分に長いことが前提になります。一般に、ブロードバンド ネットワーク（ケーブル、DSL）がこれに該当します。

スタティック サブスクライバアウェア モード

SCAS BB は、着信 IP アドレスをスタティックにバインドしてサブスクライバのグループにまとめる運用モードをサポートしています。このバインドにより、特定のサブスクライバとの間で送受信されるトラフィックを 1 つのグループとして制御（たとえば、サブスクライバとの間の P2P トラフィックを制限）できるほか、使用状況レポートをその範囲で提供できます。

スタティック サブスクライバアウェア モードは、特定の IP アドレスまたはアドレス範囲を使用するエンティティが、ダイナミックに変更されない場合に対応します。これには、次のような状況があります。

- サブスクライバの IP アドレスが DHCP、RADIUS などによってダイナミックに変化しない環境
- 1 つのサブスクライバ グループが共通の IP アドレス プールを使用する場合（たとえば、特定の CMTS、BRAS などでサービスするすべてのサブスクライバを一括して管理し、グループ全体で帯域幅を共有させる場合など）

スタティック サブスクライバは直接 SCE プラットフォームに定義することができ、外部の管理ソフトウェア（SM）は必要ありません。デバイスの CLI を使用して、サブスクライバのリスト、IP アドレス、および対応するパッケージを定義します（対話形式の設定のほか、インポート / エクスポート処理がサポートされます）。

ダイナミック サブスクライバアウェア モードと smartSUB Manager（SM）

ダイナミック サブスクライバアウェア モードでは、サブスクライバが現在使用している（IP）アドレスにダイナミックにバインドされたサブスクライバ情報（OSS ID およびサービス コンフィギュレーション）が SCE プラットフォームに入力されます。このモードでは、smartSUB Manager（SM）を使用してデバイスにサブスクライバ情報をプロビジョニングする必要があります。SM は、上記のような関連付けを維持するサーバ アプリケーションであり、この情報を SCE プラットフォームにリアルタイムでプロビジョニングします。

SM の全般的な機能

SM は最大で 500,000 のサブスクライバおよび 20 台の SCE プラットフォームをサポートします。シスコでは、展開の規模やタイプに応じて適切なプラットフォームを選択するためのサイジング ツールも提供しています。

SM は Java ベースのサーバ アプリケーションであり、サポート対象の Solaris プラットフォームにインストールできます（CD-ROM からインストール可能なバージョン）。SM の設定と管理には、CLU（コマンドライン ユーティリティ）と、モジュールのインストール時にターゲット プラットフォームにインストールされるコンフィギュレーション ファイルを使用します。

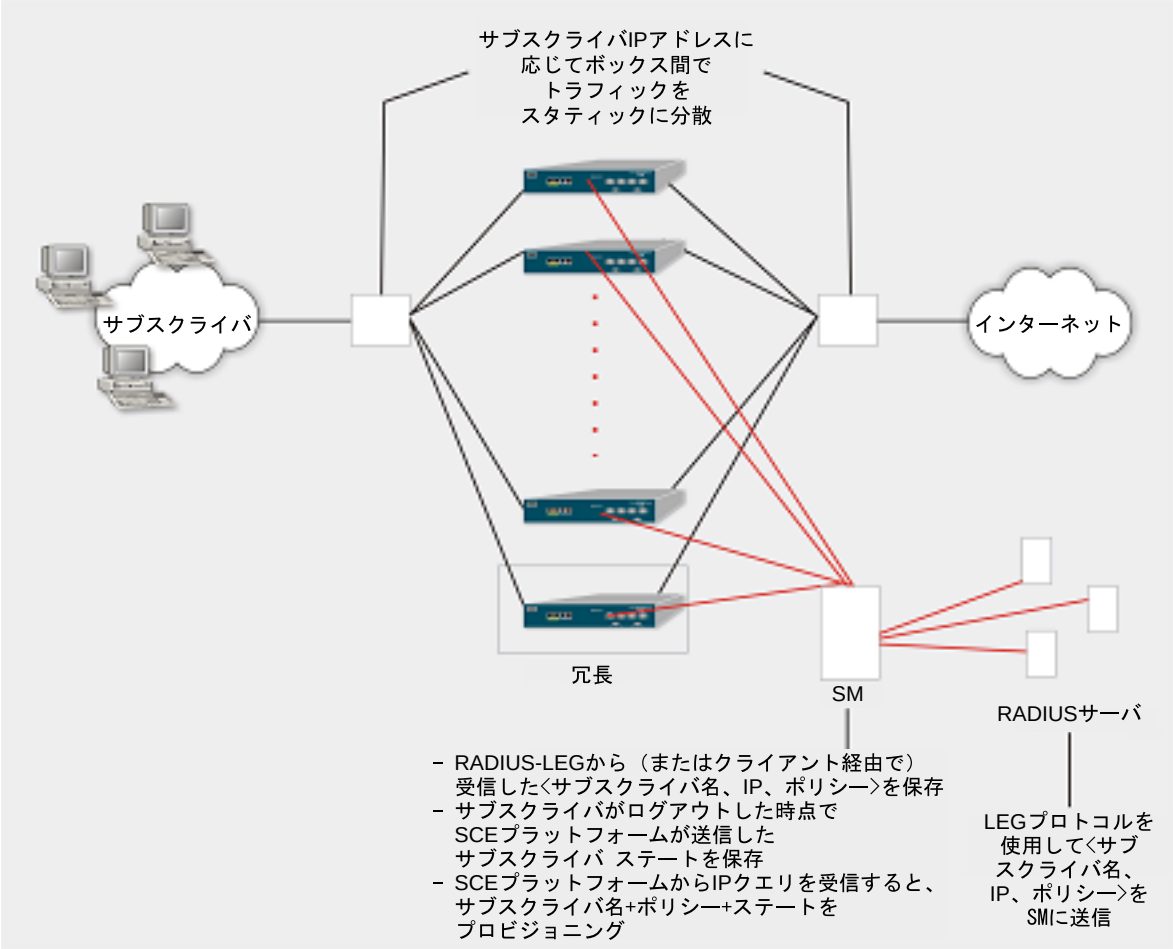
SM は、TimesTen というサードパーティ製データベース（組み込み、インメモリの市販データベース）を高性能バックエンドとして利用します。

pull モード

SMは特定のデバイス上でサブスクライバ アドレスからのネットワーク アクティビティを検出すると、SE-SM *pull モード*を利用して、SCE プラットフォームにサブスクライバ情報をダイナミックに入力します。この機能は、次のような場合に役立ちます。

- 同じリンクを使用するサブスクライバがサポートされるサブスクライバ数（40,000）より多いが、同時にアクティブなのは 40,000 以下である場合。この場合、pull モード機能を使用して、サブスクライバのアクティビティが検出されるたびに、SCE プラットフォームでサブスクライバをキャッシュイン / キャッシュアウトします。
- サブスクライバ トラフィックが実際に流れるデバイスを、IP アドレス割り当てプロセスからスタティックに類推することができないトポロジー。たとえば、次の図のように、複数の SCE プラットフォームをパラレルに配置している状況です（L3 スイッチにより、1 つの IP アドレス宛のトラフィックは常に同じパスを使用することが保証されています）。

図 6-1　ダイナミック サブスクライバアウェア（pull モード）



SCE プラットフォームはサブスクライバのわからない IP アドレスからのトラフィックを検出するたびに、SM にその情報を要求します（特定の着信 IP アドレス範囲から pull 動作を実行するようにプラットフォームを設定できます）。

SCE プラットフォームからサブスクライバが削除されると、そのサブスクライバの長期的なステート（使用済みクォータ）が後で使用できるように SM に保存されます。

このトポロジーでは、同じサブスクライバが（割り当てられる IP アドレスに応じて）そのつど異なる SCE デバイスによってサービスされる可能性があります。

また、冗長 SCE プラットフォームを準備して、いずれかのプライマリで障害が発生したときに処理を引き継ぐように設定することも可能です（N+1 冗長構成）。pull メカニズムにより、関係するサブスクライバ情報が冗長 SCE プラットフォームに入力されます。

サブスクライバ ステート

SM およびその SCE プラットフォームは、サブスクライバ ステート情報も共有します。サブスクライバがログアウトするとき（または SCE プラットフォームからキャッシュアウトされるとき）、そのサブスクライバのステート情報（クォータなど）が SM に送信され長期保存されます。そのサブスクライバが再びログインすると（またはサブスクライバに割り当てられた IP アドレスからのトラフィックが再び検出されると）、現在使用中の SCE プラットフォーム（前とは異なるデバイスの場合もある）に、この情報がプロビジョニングされます。

サブスクライバ統合：PRPC プロトコル

SM は、さまざまな環境でサブスクライバ統合を簡単に実現するために、サブスクライバ統合プロトコル（PRPC）、およびその他のツールと既成コンポーネントをサポートしています。

PRPC プロトコルは、SM へのサブスクライバ情報の伝達に使用されます。C/C++ および Java 対応の統合ツールキットが用意されています。

Java および C に対応する汎用 SM API が用意されています（『***smartSUB Manager User Guide***』を参照）。SCAS BB でこれらの API を使用する場合、ユーザはサブスクライバのパッケージ ID を表す ***packageId*** プロパティに、（SCAC 固有の）名前を指定する必要があります。

次に、SCAS BB における SM Java API の使用例を示します。

```
// subscriber-id
String subscriberName = "JerryS";

// mappings
String[] mappings     = new String[]{ "80.179.153.29" };
short[]  mappingTypes = SMApiConstants.ALL_IP_MAPPINGS;

// properties
String[] propertyKeys   = new String[]{ "packageId" };
String[] propertyValues = new String[]{ "0" };

// other settings
String  domain    = "subscribers";
boolean isMappingAdditive = false;
int     autoLogoutTime    = -1; // never

// login
smApi.login(subscriberName, mappings, mappingTypes, propertyKeys,
            propertyValues, domain, isMappingAdditive, autoLogoutTime);
```

サブスクライバ統合：CNR（DHCP）プラグイン

Cisco Network Registrar（CNR;Cisco ネットワーク レジストラ）DHCP サーバを使用するケーブル環境でサブスクライバ統合を簡単に行えるよう、シスコではそのまま使用できる CNR プラグインを提供しています。このプラグインは PRPC プロトコルを使用して、Service Control SM に IP アドレス リース情報を伝達し、CNR DHCP サーバによって CPE に割り当てられる IP アドレスに関して SM と同期を取ります。

CNR バージョンは、Windows および Solaris の両方のプラットフォームでサポートされます。

Quota Provisioning API

この章では、外部 Quota Provisioning（QP）API について説明します。

この章の内容は次のとおりです。

外部クォータ プロビジョニング

外部クォータ プロビジョニングは、Service Control パートナーがアプリケーション認識能力のあるクォータ ベースおよびボリューム ベース サービスを作成するための、クォータ（割り当て量）実施メカニズムです。このメカニズムは、外部エンティティによるサブスクライバ レベルでのクォータ プロビジョニングを可能にし、各ベンダー製のサブスクライバ管理プラットフォームとの統合を実現します。

外部クォータ プロビジョニングについての詳細は、『*Service Control Application Suite for Broadband User Guide*』を参照してください。

Quota Provisioning API（QP API）は Service Control SM API の拡張機能であり、C/C++ および Java の両方のバージョンが用意されています。QP API は、SM への接続およびサブスクライバの追加と設定に関しては SM API の機能に依存します。QP API は、クォータの設定（setSubscriberQuota）、クォータの追加（addSubscriberQuota）、およびサブスクライバのクォータ バケットにあるクォータ残量の読み取り（getRemainingSubscriberQuota）といった機能を追加します。

SM API の詳細については、『*C/C++ API for SM Guide*』および『*Java API for SM Guide*』を参照してください。

これらの機能により、OSS システムで、このようなサービス モデルをプリペイド形式やクォータに基づく消費量で使用するアプリケーション / サービスに関して、サブスクライバのトラフィック使用状況を制御するためのロジックを開発できます。

外部クォータ プロビジョニングに対応するサービス コンフィギュレーション

システムに適用する SCAS BB サービス コンフィギュレーション（PQB）を作成するとき、QP API を有効に活用するためのいくつかのガイドラインがあります。

- **パッケージ：**
  - Package Quota Management Mode（パッケージ クォータ管理モード）を「External」に設定する必要があります。
  - バケットを設定するとき、適切なバケット タイプを設定する必要があります。使用可能なタイプは、「Volume（容量、L3 KB）」または「Number of Sessions（セッション数）」です。
  - サービス ルールにおける使用量制限の定義では、適切なバケットを選択する必要があります。サービス トラフィックは、選択したバケットからクォータを消費します。ルールの違反処理アクションを使用して、バケットの使用中にトラフィックに割り当てるサービス レベルを設定できます。
- **RDR：**関連する RDR の生成をアクティブにするには、RDR Settings で次の RDR をイネーブルにする必要があります。
  - Quota Breach（クォータ違反）RDR
  - Remaining Quota（クォータ残量）RDR
  - Quota Threshold（クォータ スレッシュホールド）RDR

パッケージ、ルール、および RDR の設定についての詳細は、『*Service Control Application Suite for Broadband User Guide*』を参照してください。

クォータ バケットの状態

ここでは、サブスクライバがクォータ バケットからクォータを消費するとき、および QP API によって追加のクォータをプロビジョニングするときの、クォータ バケットのさまざまな状態について説明します。

バケットの状態としては、次の 3 種類があります。

- **スレッシュホールドより上：**バケットが**スレッシュホールドより上**の場合、クォータは消費されており、クォータ残量が Remaining Quota RDR でレポートされます。
- **スレッシュホールドより下：**クォータの残量が**スレッシュホールドより下**になると、Quota Threshold RDR が生成されます。この RDR に対応して、クォータの追加または設定を行えば、バケットを再びスレッシュホールドより上にして、枯渇を未然に防げる可能性があります。クォータの残量は、クォータの消費に応じて Remaining Quota RDR でレポートされます。
- **枯渇（Depleted）**：クォータが 0 未満になると、バケットは**枯渇**します。この場合、バケットはクォータの欠損すなわち「マイナス」のクォータ残量を維持し、これが Remaining Quota RDR でレポートされます。「枯渇」の状態になると、Quota Breach RDR が生成され、ルールの違反処理アクション（定義されている場合）がトラフィックに適用されます。この時点でクォータの追加または設定を行えば、バケットの枯渇状態を解除できる可能性があります。

初期化されていないバケットのクォータ量は 0 です。つまり、クォータが 1 回でも使用されると、バケットが枯渇します。

クォータ プロビジョニングのライフサイクル

ここでは、クォータ プロビジョニング処理と、サブスクライバがログインおよびログアウトし、ネットワーク トラフィックを発生させる各段階でのサブスクライバのクォータ状態について、そのライフサイクルを説明します。

クォータ プロビジョニング処理に関して理解しておくべき重要な点は、QP API の関連するメソッドによって要求した Add または Set などの処理が、キューに格納されて即時に結果が返されるとしても、実際にクォータが変更されるのは、サブスクライバがトラフィックを生成したとき、すなわち、サブスクライバが何らかのネットワーク アクティビティを実行したときだけであるという点です。したがって、外部のプロビジョニング システムがクォータの変更を要求してから、その変更が実際に行われるまでの間、サブスクライバの状態はクォータ変更を反映しません。そのため、この中間的な時点でサブスクライバのクォータ残量を読み取っても、表示される値には、最新の変更が組み込まれていません。

次の図で、クォータ プロビジョニング処理およびサブスクライバのネットワーク アクティビティの結果、クォータが変化する例を示します。

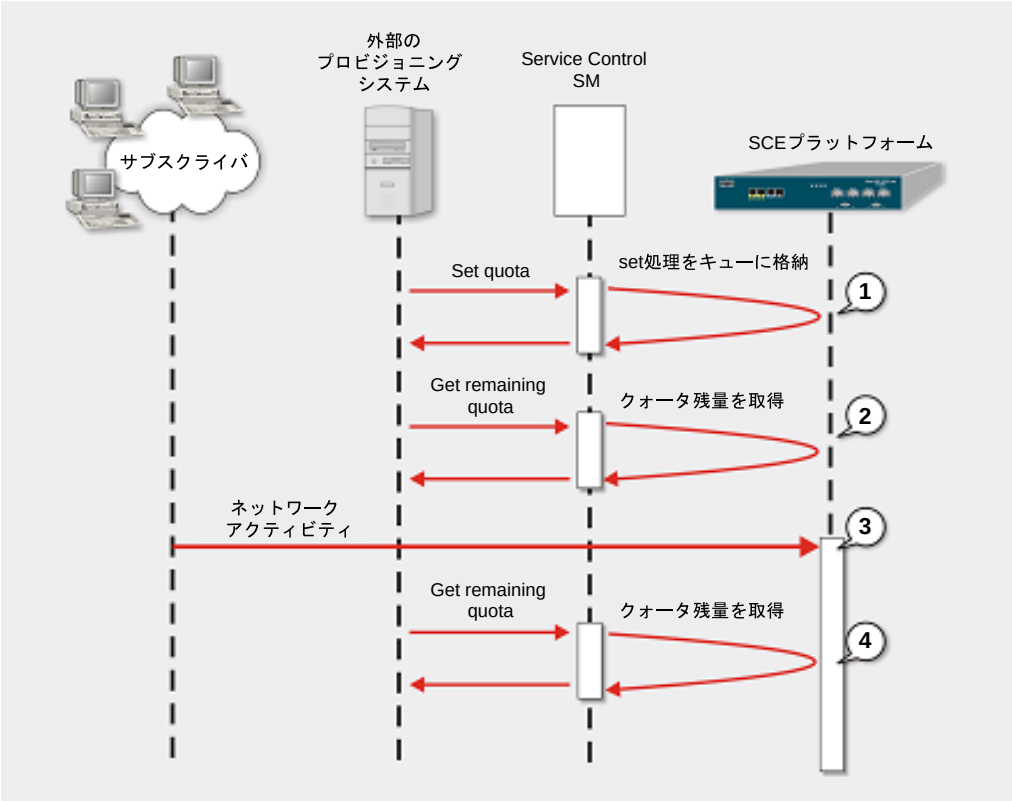
ステップ 1　外部クォータ プロビジョニング システムがクォータの変更（この例では set-quota）を実行すると、その変更はキューに格納され、まだ処理されません（図中 1）。

ステップ 2　したがって、外部クォータ プロビジョニング システムがクォータ残量を読み取ると、変更前の古い値が表示されます（図中 2）。

ステップ 3　サブスクライバがトラフィックを発生させて初めて、変更が行われます（図中 3）。

ステップ 4　このとき、外部クォータ プロビジョニング システムがクォータ残量を読み取ると、更新済みの値が表示されます（図中 4）。

図 7-1　クォータ プロビジョニングのライフサイクル



制限事項

- **プラスのクォータ残量は、バケットごとに 256 GB** という上限があります。

  この 256 GB という限界を超えてサブスクライバのクォータを増やそうとすると、エラーにはなりませんが、結果のクォータ残量がマイナスに変わります。したがって、add-quota コマンドを使用するときは注意が必要です。

  同様に、サブスクライバのクォータ欠損は、バケットごとに最大で 256 GB です。サブスクライバのいずれかのバケットが欠損しているとき、そのバケットの残量はマイナスの値（-256 GB まで）です。このマイナスの値をさらに超過すると、システムは過剰消費されたバケットへの課金を停止し、残量は -256 GB のままになります。

- **連続 256 回の set-quota 処理に関する問題**

  これも上記の問題と同様ですが、サブスクライバがログオフしているときや非アクティブのときは、set-quota コマンドによるクォータ残量の設定を頻繁に行うべきではありません。正確には、サブスクライバが非アクティブのときに set-quota コマンドで 256 回連続してクォータ バケットを更新すると、エラーにはなりませんが、バケットの残量が不正確になります。

外部 Quota Provisioning API のインストール

C および Java に対応する QP API は、それぞれ別のファイルでパッケージ化されています。このファイルの内容は次のとおりです。

- qp-c-api-dist.tar.gz
  - マニュアル
  - インクルード ファイル
  - Solaris SO ファイル
  - WinNT DLL ファイルおよび LIB ファイル
- qpapi.jar
- qpapi-javadoc.zip

インストールするには、最初に SM API をインストールしたあと、QP API を同じ手順でインストールします。

コンパイルして実行するには、SM API のコンパイルおよび実行手順に従い、QP API ファイルも *PATH/Class-path* に追加します。Java API を使用する場合は、classpath に *um_core.jar* をインクルードします。*um_core.jar* は、SCAS BB パッケージに含まれています。

QP API（Java）のメソッド

ここでは、Java 用のブロッキング QP API のメソッドを示します。各メソッドのシグニチャに続き、入力パラメータおよび戻り値を説明します。

addSubscriberQuota

構文

```
void addSubscriberQuota(String subscriberName,
                        int[] quota)
throws QPApiException, ConnectionDownException
```

説明

特定のサブスクライバのクォータ バケットの現在のクォータに、特定のクォータを追加します。

パラメータ

`subscriberName`：サブスクライバ ID

`quota`：特定のサブスクライバのクォータ バケットの現在のクォータに追加する、16 のクォータ値（L3 KB またはセッション数）の配列

戻り値

なし

addSubscriberQuota

構文

```
void addSubscriberQuota(String subscriberName,
                                int bucketNum
                            int[] quota)
throws QPApiException, ConnectionDownException
```

説明

特定のサブスクライバの特定のクォータ バケットの現在のクォータに、特定のクォータを追加します。

パラメータ

`subscriberName`：サブスクライバ ID

`bucketNum`：バケット番号

`quota`：特定のサブスクライバのクォータ バケットの現在のクォータに追加するクォータ値（L3 KB またはセッション数）

戻り値

なし

getSubscriberQuota

構文

```
int[] getSubscriberQuota(String subscriberName)
throws QPApiException, ConnectionDownException
```

説明

特定のサブスクライバの各クォータ バケットのクォータ残量を取得します。

パラメータ

`subscriberName`：サブスクライバ ID

戻り値

特定のサブスクライバの各クォータ バケットにある、クォータ残量の値（L3 KB またはセッション数）の配列

setSubscriberQuota

構文

```
void setSubscriberQuota(String subscriberName,
                        int[] quota)
throws QPApiException, ConnectionDownException
```

説明

特定のサブスクライバのクォータ バケットに、特定のクォータを設定します。

パラメータ

`subscriberName`：サブスクライバ ID

`quota`：特定のサブスクライバのクォータ バケットに設定する、16 のクォータ値（L3 KB またはセッション数）の配列

戻り値

なし

QP API（Java）のコード例

ここでは、QP API（Java）を使用したコード例を示します。

```
import java.util.Arrays;
import com.cisco.apps.scas.api.QPApiException;
import com.cisco.apps.scas.api.QPBlockingApi;
import com.cisco.management.framework.client.ConnectionDownException;
/**
 * External Quota Provisioning Example
 **/
public class ExternalQPExample {

                public static final String SM_ADDRESS = "10.1.12.65";

                static public void main(String[] args) throws Exception{

                        QPBlockingApi qpBlockingApi = null;
                        try{
                                //instantiate api and create a connection
                                qpBlockingApi = new QPBlockingApi();
                                qpBlockingApi.connect(SM_ADDRESS);

                                int[] defaultQuota = new int[16];

                              //provision each bucket of subscriber "sub1"
                              //with 1000 KBytes or 1000 Sessions.
                                Arrays.fill(defaultQuota,1000);
                                qpBlockingApi.setSubscriberQuota( "sub1",
defaultQuota);

                              //dd to bucket 2 of subscriber "sub1"
                              //with 500 KBytes or 500 Sessions.
                                qpBlockingApi.addSubscriberQuota( "sub1", 0, 500);

                               //get "sub1" remaining quota - this method
```

```
                                //invocation will work only if the
                                //subscriber is logged in.
                                //The remaining quota will reflect
                               //the last two modifications if
        //subscriber has generated traffic
                                 int[] remainingQuota =
                                              qpBlockingApi.getRemainingSu
bscriberQuota("sub1");
                          printRemainingQuota( remainingQuota);

                         }catch(QPApiException e){
                                 System.out.println("Error Code is: " + e.getCode()
);
                                 System.out.println("Error Message is: " +
e.getMessage());
                         }catch(ConnectionDownException e){
                                System.out.println("Error due to connection failure:
" + e.getMessage());
                         }catch( IllegalStateException e ){
                                System.out.println("Error due to connection failure:
" + e.getMessage());
                         }finally{
                                 if( qpBlockingApi != null &&
qpBlockingApi.isConnected() )
                                         qpBlockingApi.disconnect();
                         }

                         System.exit(0);
                 }

            private static void printRemainingQuota(int[] remainingQuota) {
                         for (int bucketIdx = 0;
                                 bucketIdx < remainingQuota.length;
                                 bucketIdx++) {
                                 System.out.println(
                                         "bucketIdx="
                                                 + bucketIdx
                                                 + ",remainingQuota="
                                               + remainingQuota[bucketIdx]);
                         }
                 }


}
```

QP API（C）のメソッド

ここでは、C に対応するブロッキング QP API のメソッドを示します。各メソッドのシグニチャに続き、入力パラメータおよび戻り値を説明します。

（注）　QP API C 関数名には「QPB_」というプレフィクスが付きます。すべての QP API C 関数で、最初のパラメータは、`argApiHandle`（`init` 関数をコールして作成される API ハンドル）です。

addQuota

構文

```
ReturnCode* QPB_addQuota (QPB_HANDLE argApiHandle, char* argName, int* argQuotas)
```

説明

特定のサブスクライバのクォータ バケットの現在のクォータに、特定のクォータを追加します。

パラメータ

`argName`：サブスクライバ ID

`argQuotas`：特定のサブスクライバのクォータ バケットの現在のクォータに追加する、16 のクォータ値（L3 KB またはセッション数）の配列

戻り値

ReturnCode 構造体のポインタ

getRemainingQuota

構文

```
ReturnCode* QPB_getRemainingQuota (QPB_HANDLE argApiHandle, char* argName)
```

説明

特定のサブスクライバの各クォータ バケットのクォータ残量を取得します。

パラメータ

`argName`：サブスクライバ ID

戻り値

特定のサブスクライバの各クォータ バケットにあるクォータ残量の値（L3 KB またはセッション数）の整数配列を格納した **`ReturnCode`** 構造体のポインタ

setQuota

構文

```
ReturnCode* QPB_setQuota (QPB_HANDLE argApiHandle, char* argName, int* argQuotas)
```

説明

特定のサブスクライバのクォータ バケットに特定のクォータを設定します。

パラメータ

`argName`：サブスクライバ ID

`argQuotas`：特定のサブスクライバのクォータ バケットに設定する、16 のクォータ値（L3 KB またはセッション数）の配列

戻り値

ReturnCode 構造体のポインタ

QP API（C）のコード例

ここでは、QP API（C）を使用したコード例を示します。

- サブスクライバのクォータの設定、クォータの追加、およびクォータ残量の取得

```
#include <stdio.h>
#include "QpApiBlocking_c.h"
void onExampleDisconnect()
{
    printf("*** DISCONNECTED ***\n");
}
int example(char* argSmAddress)
{
    // init
    printf("initializing\n");
    QPB_HANDLE api;
    api = QPB_init(10,0,20000,10,30);
    if (api == NULL) {
        printf("init failed\n");
        return -1;
    }
    QPB_setName(api,"qp-example");

    // set a disconnect-listener
    QPB_setDisconnectListener(api,onExampleDisconnect);

    // connect
    printf("connecting\n");
    int cnt = 0;
    while (QPB_connect(api,argSmAddress,14374) == false) {
        if (cnt++ > 10) {
            printf("connect failed, too many reconnects, aborting\n");
            return -1;
        }
    }


    // quota operations

    // prepare a quota bucket array [100, 200, 300,...]
    int quotas[16];
    for (int i = 0; i < 16; ++i) {
        quotas[i] = (i+1)*100;
    }

    // set the quota to subscriber "subs1"
    printf("setting quota\n");
    ReturnCode* rt;
rt = QPB_setQuota(api,(char*)"subs1",quotas);
    if (isReturnCodeError(rt)) {
        printf((char*)"set-quota failed\n");
        printReturnCode(rt);
        return -1;
    }
    freeReturnCode(rt);
// add quota to subscriber "subs2"
```

```
    printf("adding quota\n");
    rt = QPB_addQuota(api,(char*)"subs2",quotas);
    if (isReturnCodeError(rt)) {
        printf((char*)"add-quota failed\n");
        printReturnCode(rt);
        return -1;
    }
   freeReturnCode(rt);

    // getting remaining quota of subscriber "subs3"
    // this method invocation will work only if the subscriber is logged in.
    // the remaining quota will reflect recent modifications if the
    // subscriber has generated traffic
    printf("getting remaining quota\n");
    rt = QPB_getRemainingQuota(api,(char*)"subs3");
    printReturnCode(rt);
    if (isReturnCodeError(rt)) {
        printf("get-remaining-quota failed\n");
        return -1;
    }
    freeReturnCode(rt);


    // disconnect
    if (QPB_disconnect(api) == false) {
        printf("disconnect failed\n");
        return -1;
    }
    QPB_release(api);

    return 0;
}


int main(int argc, char* argv[])
{
    char* smAddress = (char*)"10.1.12.82";
    printf("SM address: %s\n",smAddress);

    if (example(smAddress) < 0) {
        printf("example failed\n");
        return -1;
    }
    return 0;
}
```

エラー コードと例外処理

Quota Provisioning API のエラー コード

次の表に、Quota Provisioning API で発生する可能性のあるエラー コードを示します。Quota Provisioning API を SM API と併用する場合は、後者のエラー コードが表示されることもあります。そのようなエラー コードの詳細は、『*smartSUB Programmer's Guide*』の *付録 A* を参照してください。

表 7-1　Quota Provisioning API のエラー コード

エラー コード	説明	推奨する措置 / 注意事項
40,000	**不正な引数による例外**：指定したクォータの値が不正です（たとえば、値が大きすぎる）。	引数に指定したすべてのクォータ値が有効かどうかを確認します。
40,002	**ログインしていないサブスクライバに関する例外**：ログインしていないサブスクライバに関して処理を実行しようとしました。	サブスクライバが SCE デバイスにログインしてから再度実行してください。
40,003	**不明の例外**：予期されないエラーが発生しました。	カスタマー サポートに連絡してください。
40,004	**タイムアウト例外**：SM データベースが長期にわたってロックされた場合に発生します（内部エラー）。	カスタマー サポートに連絡してください。
40,030	**非アクティブなサブスクライバに関する例外**：サブスクライバのコンテキストが予想外に見つからない場合に発生します（内部エラー）。	カスタマー サポートに連絡してください。

いずれのエラーも、すべての QP API メソッドで発生するわけではありません。

Java API での例外の管理

Java API で発生する例外は、次の 3 タイプです。

- **ConnectionDownException**：SM への確立済みの接続が、処理の途中で切断された場合に発生します。
- **IllegalStateException**：SM への接続が突発的に切断された場合に発生することがあります。
- **QPApiException**：その他のすべてのエラー。

QPApiException が発生した場合は、qpApiException.getCode() を使用し、上記の（または SM のマニュアルの *付録 A* にある）表の「エラー コード」欄と比較してください。qpApiException.getMessage() を使用して、ストリング エラー メッセージを受信します。

C/C++ API でのエラー コードの管理

関数コールが失敗すると、エラー コードが返されます。そのエラー コードを、上記の（または SM のマニュアルの *付録 A* にある）表の「エラー コード」欄と比較してください。

Reporter CLI

この章では、Service Control Reporter CLI（コマンドライン インターフェイス）を使用して Service Control reporter アプリケーションを実行するさまざまな方法について、構文、スイッチ、およびオプションも含めて説明します。

この章の内容は次のとおりです。

Reporter CLI の概要

Reporter CLI は、SCASl Reporter GUI の機能を補足するコマンドライン アプリケーションをベースとしています。この CLI は*実行可能プログラム*として、または *CGI スクリプト*として実行できます。Service Control Reporter CLI は、対応する SCAS Reporter GUI を超える機能性と柔軟性を提供し、サードパーティ製アプリケーションと統合することにより、使用状況および統計に基づく出力の生成が可能です。

構文と使用法

SCAS Reporter は、次の方法で実行できるアプリケーションです。

- **コマンドライン** ― コマンドライン バージョンでは、パラメータの入力が受け付けられます。
- **コマンド ファイル** ― コマンド ファイル バージョンは、コマンドライン パラメータの入ったファイルを入力として読み込みます。バッチ ファイルとして実行できます。
- **起動される実行可能プログラム** ― 起動元のアプリケーションにプログラムを埋め込みます。アプリケーションは、SCAS Reporter にパラメータとして渡されるストリングをダイナミックに構築します。

さらに、次の場所から SCAS Reporter を呼び出すことができます。

- **CGI BIN ディレクトリ**、**ISAPI ブラウザ**、および **NSAPI ブラウザ** ― SCAS Reporter は、CGI BIN ディレクトリからでも、ISAPI または NSAPI 準拠のブラウザからでも呼び出せます。アプリケーションの出力は HTML 形式で返され、ブラウザで表示されます。エラーが検出された場合は、該当するエラー メッセージが表示されます。

コマンドラインでの使用法

SCAS Reporter アプリケーションを起動するためのコマンドライン構文は、次のとおりです。

```
reporter { [-r report-id] | [-n report-name] | [-i report-index] }
     -f [report-format] -k key=value -e [CON]
     -l user-name\password@host-machine
     [drive:][path] <report-filename>
```

コマンドラインでの構文

SCAS Reporter アプリケーションを起動するときコマンドラインで使用するスイッチは、次のとおりです。

- スイッチ `-f`、`-k`、および `-e` は省略可能です。
- `-r`、`-n`、または `-i` スイッチのうち、いずれか 1 つだけを指定する必要があります。
- その他のパラメータは必須です。

コマンドライン オプション

SCAS Reporter アプリケーションを起動するときにコマンドラインで使用するオプションについて、次の表で説明します。

表 8-1　SCAS Reporter アプリケーションのコマンドライン オプション

オプション	SCAS Reporter によるアクション	説明
-r <*report-id*>	report-id を使用してレポートを生成します。	このパラメータは、定義済みのテンプレートを使用してレポートを生成する場合に使用します。
-n <*report-name*>	report-name を使用してレポートを生成します。	このパラメータは、個人用のテンプレートを使用してレポートを生成する場合に使用します。指定した名前は、保存されているレポート名と照合されます。レポート名には大文字と小文字の区別があります。
-i <*report-index*>	report-index を使用してレポートを生成します。	このパラメータは、個人用のテンプレートを使用してレポートを生成する場合に使用します。
-f [report-format]	指定したレポート フォーマットで、グラフまたは表としてレポートを生成します。report-format を省略すると、デフォルトのフォーマットが使用されます。アスタリスク付きのフォーマットが、デフォルトです。 表のフォーマット： JPG GIF * HTM グラフのフォーマット： CSV* HTM XLS	
-k key=value	指定する値で定義済みのキーを上書きします。	
-l user-name\password @hostname	（必須）user-name\password@hostname machine を使用してログインします。	Reporter の実行可能プログラムをホスティングしているワークステーションのユーザ名およびパスワードを使用してください。
<*report-filename*>	（必須）出力の転送先ファイル名。	ファイル名がすでに存在する場合、元のファイルが上書きされます。
-e [CON]	出力（メッセージおよびエラー）をエラー メッセージ ファイルにリダイレクトします。オプションのCONスイッチとしてコンソール ウィンドウを使用し、ウィンドウが開いている場合、そのウィンドウにエラー メッセージがリダイレクトされます。	Reporter は Win32 アプリケーションなので、stderr はエラー メッセージ ファイルにリダイレクトされます。

コマンド ファイルでの使用法

コマンド ファイルから SCAS Reporter アプリケーションを起動するためのコマンドは、次のとおりです。

```
reporter@[drive:][path][command-file]
```

コマンド ファイルでの構文

コマンド ファイルから SCAS Reporter アプリケーションを起動するための構文は、次のとおりです。

- 実行可能プログラム名の後ろに、アットマーク（@）、ドライブおよびパス（省略可能）、および入力に使用するコマンド ファイルの名前（必須）を指定します。
- コマンド ファイルの行ごとに、レポートを生成するための異なるコマンドを指定します。
- セミコロン（;）で始まる行は注釈行とみなされ、無視されます。
- バックスラッシュ（\）を使用して長い行を分割できます。最終行を除く各行の最後にあるバックスラッシュが連結されます。

C

CLI（コマンドライン インターフェイス） SCE プラットフォームの管理インターフェイスの 1 つ。Telnet セッションでアクセスすることも、SCE プラットフォームの前面パネルにあるコンソール ポート経由で直接アクセスすることもできます。

Collection Manager（CM） SCE プラットフォームから RDR を収集して処理し、レポート用に準備するソフトウェア アプリケーション。

P

PQI（Service Control インストレーション）ファイル SCE プラットフォームまたは関連ソフトウェアのモジュールにインストールするアプリケーション パッケージ ファイル。

R

RDR（Raw Data Record） トラフィックで発生したイベントをレポートする目的で、SCE プラットフォームによって生成されるデータ レコード。SCE プラットフォームで生成された RDR は Collection Manager に送信され、Collection Manager データベースに保存されるか、またはサードパーティ製システムに転送されます。クォータ（「クォータ」を参照）要求を含む RDR や、サービス使用状況をレポートする RDR が一般的です。

S

SCAS BB Console SCAS システムを制御するためのユーザ インターフェイス。サービス コンフィギュレーションの作成、変更、および適用を行います。

SCE（Service Engine）プラットフォーム サービス コントロール用に特化した専用ネットワーク エレメント。このハードウェア デバイスは、ワイヤ速度でのディープ パケット解析、およびビジネス ポリシーに基づくサブスクライバの トラフィック制御が可能です。

Service Control アプリケーション SCE プラットフォームの運用方法を定義する SML プログラム。

SLI（SML Loadable Image）ファイル SLI ファイルは、SCE プラットフォームにロードされる SML アプリケーションを含んだソフトウェア パッケージです（SCAS ソリューションの一部）。この SML アプリケーションにより、SCE プラットフォームの動作が決まります。同じ POP に存在する SCE プラットフォームでも、プラットフォームごとに異なる SML アプリケーションを使用できます（オペレータは SLI ファイルにアクセスする必要はありません）。

smartSUB Manager（SM） サブスクライバ情報とサービス コンフィギュレーションのダイナミック バインディングが必要な場合に使用するミドルウェア ソフトウェア コンポーネント。SM はサブスクライバ情報を管理し、複数の SCE プラットフォームにリアルタイムでプロビジョニングします。SM はサブスクライバのサービス コンフィギュレーション情報を内部に保存し、AAA システム（RADIUS、DHCP など）と SCE プラットフォームの間のステートフル ブリッジとして動作します。

あ

アップストリーム トラフィック サブスクライバ側から SCE プラットフォームに着信するトラフィック。

アノニマス サブスクライバ モード ソリューションのモードの 1 つ。システムがトラフィックをモニタし、サブスクライバ側で使用される個々の IP アドレスに基づいて、自動的にサービス コンフィギュレーションを割り当てます。このモードを使用すると、システムを OSS システムと統合せずに、サブスクライバのトラフィックを匿名で制御できます。このモードでは、システムで定義されるサブスクライバは匿名であり、IP アドレスまたは VLAN ID によってのみ区別されます。

い

インライン接続モード SCE プラットフォームがデータ リンク上のサブスクライバ側とネットワーク側の中間に物理的に位置します。トラフィックの送受信が可能であり、トラフィック制御のほかにモニタリングも実行できます。

く

クォータ 帯域幅、容量など、特定のメトリックでの（サブスクライバの）制限。

グローバル コントローラ グローバル コントローラは、全サブスクライバを対象に、選択されたプロトコルまたはパッケージの帯域幅パーセンテージをすべて制御するために使用します。サブスクライバ BW コントローラも参照してください。

さ

サービス サービス プロバイダーがアクセス ネットワークのトップでサブスクライバに提供する付加価値項目。

プロバイダーがサブスクライバに提供するこの種の営利サービスでは、Encharge ソリューションでそれぞれ対応するサービスが定義され、サービスに関連するネットワーク トランザクションの分類や識別、使用状況のレポート、ビジネス ポリシーに基づいたトラフィックの制御が行われます。

サービス コンフィギュレーション Encharge ソリューションにおけるサービスの定義。ネットワーク トランザクションを、対応するサービスおよび SCE プラットフォームが実行すべき動作にマッピングします。サービス コンフィギュレーションには、サービス、パッケージ、帯域幅コントローラ、フィルタ ルールなどの定義が含まれます。

サービス ルール パッケージのサービス ルールを定義することで、サービスをパッケージに割り当てます。

サブスクライバ SCAS ソリューションでサービス コンフィギュレーションを実施したり、使用状況をモニタしたりする管理対象のエンティティを表す一般用語。サブスクライバは個々の IP アドレスで定義することも、IP アドレスの範囲または VLAN で定義することも可能です。システムがサポートする運用モードとしては、サブスクライバレス モード（すべての制御をグローバルに実行する）、アノニマス サブスクライバ モード、ダイナミックおよびスタティック サブスクライバアウェア モードがあります。

サブスクライバ BW コントローラ（帯域幅コントローラ） サブスクライバ帯域幅コントローラ（BW コントローラ）は、個々のサブスクライバのトラフィック帯域幅を制御します。グローバル コントローラ（p.4-9）も参照してください。

サブスクライバ起動トランザクション サブスクライバのホストによって起動されるトランザクション。

サブスクライバレス モード ソリューションのモードの 1 つ。統合を必要としないので、SM コンポーネントが不要です。このモードはサブスクライバ数や着信 IP アドレス数に影響されないので、SCE プラットフォームの点では、モニタ対象のリンクを利用するサブスクライバの総数に制限はありません。グローバル デバイス単位での制御機能やレベル解析機能しか必要とされないサイトで、このモードを使用します。

し

時間ベース ルール トラフィック ルールまたはサービス ルールのどちらにも適用できる付加価値サービス ルール。このルールは、サービス ルール テーブルのサブルールとしてリストされます。ユーザ側で定義する週時間枠のいずれか 1 つに適用します。

シグニチャ プロトコルを一意に識別するパラメータの集合。

受信専用接続モード SCE プラットフォームがデータ リンク上に物理的に存在しないので、データは受信だけで送信は不可能です。

このモードでは、トラフィック モニタリングだけが機能します。

す

スタティック サブスクライバアウェア モード 各サブスクライバに特定の IP アドレスをバインドするモード。このモードは、エンタープライズ カスタマーを制御する場合や、あらかじめ定義されたサブネットのサブスクライバ グループ（特定の CMTS/BRAS のユーザなど）を制御する場合に役立ちます。

せ

セッション（別名トランザクション） ネットワーク ホスト間の通信のインスタンス。セッションの正確な定義は、アプリケーション プロトコル（レイヤ 7）に依存します。

た

ダイナミック サブスクライバアウェア モード サブスクライバがネットワークにログオンして IP アドレスを割り当てられた時点で、IP アドレスに実際のサブスクライバ ID が対応付けられるモード。このモードで運用するには、サブスクライバに IP アドレスを割り当てる OSS システム（通常は RADIUS または DHCP ベース）に、システムを統合する必要があります。

ダイナミック シグニチャ 稼働中のアプリケーションにロード可能なシグニチャ。いったんロードすると、アプリケーションはそのシグニチャに対応付けられたプロトコルを識別するようになります。

ダウンストリーム トラフィック ネットワーク側から SCE プラットフォームに着信する（サブスクライバ宛）トラフィック。

と

トラフィック ディスカバリ レポート トランザクションの使用状況レコードに基づく、ネットワーク アクティビティに関する統計レポート。

トランザクション（別名セッション） アプリケーションによって認識され、L3、L4、または L7 特性に応じて区別される、トラフィックのイベント。プロトコルごとに異なるトランザクション タイプが存在します。

ね

ネットワーク起動トランザクション ネットワーク側のホストからサブスクライバ宛に起動されるトランザクション。

は

パッケージ 各種サービスへのアクセス レベル、課金パラメータ、および特定のイベントが発生したときに実行すべきトラフィック制御アクションを定義した、ビジネス ポリシー ルールの集合。割り当てられるパッケージ（プラン）により、そのサブスクライバのネットワーク トランザクションの制御方法と課金方法が決まります。

ふ

フィルタ ルール サービス コンフィギュレーションの一部。レイヤ 3 およびレイヤ 4 のプロパティに基づいて、SCE プラットフォームにある種のトランザクションを無視するように指示し、ソリューション サービスを回避してそのまま伝送されるようにします。

も

モニタリング レポート SCAS Reporter がサブスクライバ、パッケージ、およびグローバルの各レベルで生成する、帯域幅、容量、およびセッションの使用状況レポート。

り

リアルタイム サブスクライバ使用状況モニタリング SCE デバイスがサブスクライバを詳細にモニタし、使用状況に関する情報を頻繁にレポートすることで、詳細なレポートを簡単に生成できます。

リスト サービスの定義に使用する、IP アドレス範囲または Web アドレスのリスト。

A

C

G

J

P

Q

R

S

T

あ

い

え

か

き

く

こ

さ

し

す

せ

た

て

と

ね

は

ひ

ふ

ま

も

り

る

れ

ろ