



## **Cisco SCA BB Service Configuration API Programmer Guide**

Release 3.5.5  
July 23, 2009

**Americas Headquarters**  
Cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, CA 95134-1706  
USA  
<http://www.cisco.com>  
Tel: 408 526-4000  
800 553-NETS (6387)  
Fax: 408 527-0883

Text Part Number: OL-7207-05

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CCDE, CCENT, CCSI, Cisco Eos, Cisco HealthPresence, Cisco Ironport, the Cisco logo, Cisco Lumin, Cisco Nexus, Cisco Nurse Connect, Cisco Stackpower, Cisco StadiumVision, Cisco TelePresence, Cisco Unified Computing System, Cisco WebEx, DCE, Flip Channels, Flip for Good, Flip Mino, Flip Video, Flip Video (Design), Flipshare (Design), Flip Ultra, and Welcome to the Human Network are trademarks; Changing the Way We Work, Live, Play, and Learn, Cisco Store, and Flip Gift Card are service marks; and Access Registrar, Aironet, AsyncOS, Bringing the Meeting To You, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, CCSP, CCVP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Collaboration Without Limitation, EtherFast, EtherSwitch, Event Center, Fast Step, Follow Me Browsing, FormShare, GigaDrive, HomeLink, Internet Quotient, IOS, iPhone, iQuick Study, IronPort, the IronPort logo, LightStream, Linksys, MediaTone, MeetingPlace, MeetingPlace Chime Sound, MGX, Networkers, Networking Academy, Network Registrar, PCNow, PIX, PowerPanels, ProConnect, ScriptShare, SenderBase, SMARTnet, Spectrum Expert, StackWise, The Fastest Way to Increase Your Internet Quotient, TransPath, WebEx, and the WebEx logo are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0907R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

*Cisco SCA BB Service Configuration API Programmer Guide*  
© 2009 Cisco Systems, Inc. All rights reserved.



# CONTENTS

## **Preface** v

- Introduction v
- Document Revision History v
- Organization vi
- Related Publications vi
- Conventions vii
- Obtaining Documentation and Submitting a Service Request viii

---

## **CHAPTER 1**

### **SCA BB Service Configuration API Overview** 1-1

- Introduction 1-1
- Service Configurations 1-1
- Service Configuration API 1-1
  - The Service Configuration Management API 1-2
    - Service Configuration Management API Classes 1-3
  - The Service Configuration Editing API 1-3
    - Service Configuration Editing API Data Types 1-4
  - Program Workflow Using the Service Configuration API 1-4

---

## **CHAPTER 2**

### **Getting Started** 2-1

- Introduction 2-1
- System Requirements 2-1
- Installation Package 2-1
  - Distribution Content 2-2
  - Installing the Package 2-2
- Compiling and Running the Service Configuration API 2-2
  - Running the Service Configuration API with Java 1.5 2-2
- Service Control Engine Platform Setup 2-3
  - SCA BB Setup 2-3
  - PRPC Server 2-3

---

## **CHAPTER 3**

### **Programming with the SCA BB Service Configuration API** 3-1

- Introduction 3-1
- Service Configuration API Packages 3-1

- Package com.cisco.scabb.servconf.mgmt 3-2
- Class SCABB 3-2
  - Class SCABB Methods 3-2
    - login 3-3
    - login 3-3
    - logout 3-4
    - addNotificationListener 3-4
    - removeNotificationListener 3-4
    - getDefaultProtocolFamilies 3-5
- Class ConnectionApi 3-5
  - Class ConnectionApi Methods 3-5
    - isConnected 3-5
- Class ImportExportApi 3-5
  - Class ImportExportApi Constructor 3-6
    - ImportExportApi 3-6
  - Class ImportExportApi Methods 3-6
    - exportServiceConfiguration 3-7
    - importServiceConfiguration 3-7
    - importFlavors 3-8
    - importFlavors 3-8
    - importZones 3-9
    - importZones 3-9
    - importProtocols 3-10
    - importProtocols 3-10
    - importServices 3-11
    - importServices 3-11
    - exportProtocols 3-12
    - exportProtocols 3-12
    - exportZones 3-13
    - exportZones 3-13
    - exportFlavors 3-14
    - exportFlavors 3-14
    - exportServices 3-15
    - exportServices 3-15
    - loadListArray 3-16
- Class ServiceConfigApi 3-16
  - Class ServiceConfigApi Methods 3-16
    - applyServiceConfiguration 3-17
    - applyServiceConfiguration 3-17

applyServiceConfiguration	3-18
retrieveServiceConfiguration	3-19
updateValuesIni	3-19
updateValuesIni	3-20
validateServiceConfiguration	3-20
importServConf	3-21
exportServConf	3-21
importDefaultServConf	3-22
Class ServiceConfig	3-22
Class ServiceConfig Methods	3-22
getCalendarList	3-23
getClassificationCfg	3-23
getDynamicSignatureScript	3-23
getPackageList	3-24
getPolicySettings	3-24
getProtocolList	3-24
getProtocolRedirectIndexNameArray	3-24
getProtocolRedirectString	3-25
getRealTimeFrameName	3-25
getServiceList	3-26
getSubNotifications	3-26
getTimeFrameNames	3-26
getProtocolRedirectString	3-27
getZoneList	3-27
isProtocolRedirectable	3-28
setProtocolRedirectString	3-28
setProtocolRedirectString	3-29
setTimeFrameName	3-29
setTimeFrameName	3-30
setTimeFrameNames	3-30
Service Configuration API Programming Guidelines	3-30
Connections to the SCE Platform	3-30
Service Configuration API Code Examples	3-31
Applying a Service Configuration	3-31
Updating Zones Automatically	3-33
Listing Names of Services and Packages	3-36

Service Configuration API Client Logging 4-1



## Preface

---

Revised: July 23, 2009, OL-7207-05

## Introduction

This preface describes who should read the *Cisco SCA BB Service Configuration API Programmer Guide*, how it is organized, and its document conventions.

This guide is for Java developers responsible for integrations involving tools that automatically configure the Service Control Application running on a Service Control Engine (SCE) platform.

## Document Revision History

The Document Revision History below records changes to this document.

**Table 1**      **Document Revision History**

<b>Revision</b>	<b>Cisco Service Control Release and Date</b>	<b>Change Summary</b>
OL-7207-05	Release 3.5.5 July 23, 2009	Versioned document to Release 3.5.5
OL-7207-04	Release 3.1.5 January 2008	Updated document for Release 3.1.5
OL-7207-03	Release 3.1.0 May 2007	Updated document for Release 3.1.0
OL-7207-02	Release 3.0.0 December 2005	This document was completely rewritten for version 3.0 of the Cisco Service Control Application for Broadband (SCA BB). The document title was also modified.
OL-7207-01	Release 2.5.5 February 2005	Created the <i>Service Control Application Suite for Broadband API Programmer's Guide</i> .

# Organization

The major sections of this guide are as follows:

**Table 2**      **Documentation Organization**

Chapter	Title	Description
Chapter 1	<a href="#">SCA BB Service Configuration API Overview</a>	Describes the various terms and concepts that are used when working with the Cisco SCA BB Service Configuration API (Service Configuration API).
Chapter 2	<a href="#">Getting Started</a>	Explains how to install, compile, and run the Service Configuration API.
Chapter 3	<a href="#">Programming with the SCA BB Service Configuration API</a>	Provides a detailed description of the Service Configuration API programming structure and classes, and of methods of the main Service Configuration Management API class, and also contains programming guidelines and code examples.
Chapter 4	<a href="#">Logging and Troubleshooting</a>	Describes the Service Configuration API logging functionality that allows you to troubleshoot the Service Configuration API integration.

## Related Publications

The following publications are available for the Cisco Service Control Application for Broadband:

- [Cisco Service Control Application for Broadband Reference Guide](#)
- [Cisco Service Control Application for Broadband User Guide](#)
- [Cisco SCMS SCE Subscriber API Programmer Guide](#)
- The SM API programmers guides:
  - [Cisco SCMS SM Java API Programmer Guide](#)
  - [Cisco SCMS SM C/C++ API Programmer Guide](#)
- [Cisco Service Control Management Suite Subscriber Manager User Guide](#)

# Conventions

This document uses the following conventions:

**Table 3**      **Documentation Conventions**

Convention	Description
<b>boldface font</b>	Commands and keywords are in <b>boldface</b> .
<i>italic font</i>	Arguments for which you supply values are in <i>italics</i> .
[ ]	Elements in square brackets are optional.
{x   y   z}	Alternative keywords are grouped in braces and separated by vertical bars.
[x   y   z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string, or the string will include the quotation marks.
screen font	Terminal sessions and information that the system displays are in <code>screen font</code> .
<b>boldface screen font</b>	Information you must enter is in <b>boldface screen font</b> .
<i>italic screen font</i>	Arguments for which you supply values are in <i>italic screen font</i> .
<>	Nonprinting characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.



**Note**

Means *reader take note*. Notes contain helpful suggestions or references to materials not covered in this manual.



**Caution**

Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.



**Warning**

Means *reader be warned*. In this situation, you might do something that could result in bodily injury.

## Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

<http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html>

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.



# CHAPTER 1

## SCA BB Service Configuration API Overview

---

Revised: July 23, 2009, OL-7207-05

### Introduction

This chapter describes various terms and concepts that are utilized when working with the Cisco SCA BB Service Configuration API.

- [Service Configurations, page 1-1](#)
- [Service Configuration API, page 1-1](#)

### Service Configurations

One of the fundamental entities in the Cisco Service Control solution is a service configuration . A service configuration is a collection of configuration parameters that together determine how the Cisco Service Control Application, which runs on the Service Control Engine (SCE) platform, performs classification, accounting and reporting, and control of network traffic.

By editing a service configuration and applying it to the SCE platform, you can change the way traffic is classified and enforce different policies.

For more information about service configurations and traffic processing, see the [Cisco Service Control Application for Broadband User Guide](#).

### Service Configuration API

Usually, editing and managing service configurations is done manually in the Cisco SCA BB Console, which is a GUI tool that lets you edit every aspect of the configuration and apply changes to the SCE platform.

The Service Configuration API provides the ability for external applications to change the policy that is enforced by an SCE platform, by programmatically editing service configurations and applying these configurations to SCE platforms.

The Service Configuration API has two parts, each dealing with a different type of task:

- The Service Configuration Management API—Handles the tasks that are performed on a service configuration as a whole, such as applying a configuration to an SCE platform, and saving the configuration to a file
- Service Configuration Editing API—Handles changes to a single service configuration, such as changing a classification criterion or adding a policy rule
- [The Service Configuration Management API, page 1-2](#)
- [The Service Configuration Editing API, page 1-3](#)
- [Program Workflow Using the Service Configuration API, page 1-4](#)

## The Service Configuration Management API

The Service Configuration Management API can be used to perform the following tasks:

- Create a new service configuration—Creates a new service configuration object, with default configuration values. This configuration can be edited, and then applied to an SCE platform or saved to a file.
- Save a service configuration to a file—Saves a service configuration to a PQB file. PQB files created by the API can be edited in the SCA BB Console, and vice versa.
- Read a service configuration from a file—Loads a PQB file, before applying it to an SCE platform or for editing purposes.
- Apply a service configuration to an SCE platform—Activates a service configuration on an SCE platform (the main operation of the API). This task creates a connection from the API program to the SCE platform and then applies the service configuration to the SCE platform over the connection.
- Retrieve a service configuration from an SCE platform—Gets the currently applied service configuration from the SCE platform. This task creates a connection from the API program to the SCE platform and then copies the configuration from the SCE platform over the connection.
- Export parts of the service configuration to CSV files—Exports parts of a service configuration, such as zones, flavors, and protocols, to a CSV file. CSV files can be manipulated manually or by some external system, and then imported back into service configurations.
- Import parts of the service configuration from CSV files—Imports parts of a service configuration from a CSV file to an open service configuration. A program can create such CSV files automatically, import them into a service configuration, and then apply the updated configuration to the SCE platform. Those parts of service configurations that can be imported to and exported from CSV files are described in the “[Using the Service Configuration Editor](#)” chapter of the *Cisco Service Control Application for Broadband User Guide*. The CSV formats are described in the “[CSV File Formats](#)” chapter of the *Cisco Service Control Application for Broadband Reference Guide*.

Those parts of service configurations that can be imported to and exported from CSV files are described in the “[Using the Service Configuration Editor](#)” chapter of the *Cisco Service Control Application for Broadband User Guide*. The CSV formats are described in the “[CSV File Formats](#)” chapter of the *Cisco Service Control Application for Broadband Reference Guide*.

## Service Configuration Management API Classes

The main classes of the Service Configuration Management API are:

- SCABB—Provides methods for connecting to the SCE platform, to be used for apply and retrieve operations
- ConnectionApi—The connection to the SCE platform
- ImportExportApi—Provides methods for saving service configurations to and reading service configurations from PQB files, and for importing and exporting parts of service configurations to and from CSV files
- ServiceConfigApi—Provides the methods for creating a new service configuration, and for applying service configurations to and retrieving service configurations from SCE platforms
- ServiceConfig—Objects of this class are containers of configuration parameters that, when applied to the SCE platform, determine how the service control application performs classification, accounting and reporting, and control over network traffic

## The Service Configuration Editing API

The Service Configuration Editing API can be used to perform the following tasks:

- Inspecting a service configuration—The API can inspect a configuration that is retrieved from an SCE platform to determine, for example, which services and subscriber packages it contains. This can be used for comparing two configurations.
- Configuring criteria for traffic classification—A service configuration can determine how traffic is classified according to various criteria, for example, according to the URL of HTTP transactions. For example, the API can be used to automatically update lists of URLs that deny access to certain websites.
- Configuring various aspects of traffic control—The service configuration determines how network traffic is controlled using rules per service and per subscriber package, elaborate BW control hierarchy, and quota management models. The API can be used to change these settings, thereby changing the policy enforced by the SCE platform in an automatic manner, for example, as a response to some external trigger or at specified time intervals.

**Note**

---

Provisioning policy to subscribers cannot be performed using this API. Use the Subscriber APIs described in the SM API programmer's guides (see the [Cisco SCMS SM Java API Programmer Guide](#) or the [Cisco SCMS SM C/C++ API Programmer Guide](#)).

---

**Note**

---

Monitoring counters cannot be performed using this API. Monitor counters via SNMP to the SCE platform (see the “[SCA BB Proprietary MIB Reference](#)” chapter of the [Cisco Service Control Application for Broadband Reference Guide](#)) or via SQL to the Cisco Service Control Management Suite Collection Manager database (see the [Cisco Service Control Management Suite Collection Manager User Guide](#)).

---

## Service Configuration Editing API Data Types

- The main data types for traffic classification settings are services, protocols, signatures, zones, and flavors.
- The main data types for traffic accounting and reporting are counter definitions and various RDR parameters.
- The main data types for traffic control are packages, rules, subscriber BW controllers and global BW controllers.

An explanation of the objects (service configuration entities) that make up a service configuration is beyond the scope of this document: refer to the “[Traffic Processing Overview](#)” chapter of the *Cisco Service Control Application for Broadband User Guide*. Using the SCA BB Console (which uses the Service Configuration Editing API) is a good way to become familiar with these configuration objects, and to get an idea of what you can do with the API.

## Program Workflow Using the Service Configuration API

A typical workflow for an application that uses the Service Configuration API is:

1. The application retrieves a service configuration from the SCE platform.
2. The application modifies the service configuration.

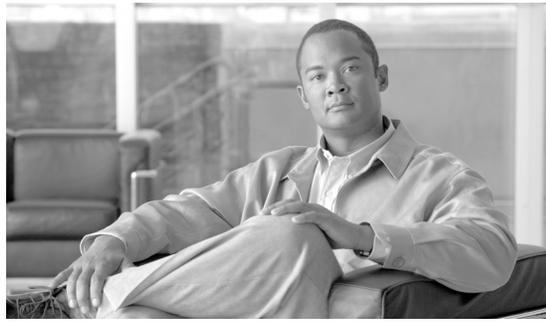
For example, the application may import zone settings from a CSV file (a zone is a list of network-side IP addresses that serve as a classification criterion). After a zone is defined in the service configuration, you can create a rule to control all network traffic going to that zone.

See the code example [Updating Zones Automatically, page 3-33](#).

3. The application applies the modified service configuration to the SCE platform so that classification and enforcement can take place according to the updated configuration.

See the code example [Applying a Service Configuration, page 3-31](#).

Another common workflow makes use of a set of predefined service configurations (created manually in the SCA BB Console). In this workflow, the application uses the API to apply each service configuration to the SCE platform, either as a response to some external trigger or at specified times.



# CHAPTER 2

## Getting Started

---

Revised: July 23, 2009, OL-7207-05

### Introduction

This chapter explains how to install, compile, and run the Cisco SCA BB Service Configuration API.

- [System Requirements, page 2-1](#)
- [Installation Package, page 2-1](#)
- [Compiling and Running the Service Configuration API, page 2-2](#)
- [Service Control Engine Platform Setup, page 2-3](#)

### System Requirements

You can run the Service Configuration API on Win32, Solaris, and Linux platforms.

Compiling and running Java programs using the Service Configuration API requires JDK and JRE versions 1.4 or 1.5.



**Note**

---

To use the API with Java 1.5, you must set a special JRE option (see [Running the Service Configuration API with Java 1.5, page 2-2](#)).

---

### Installation Package

The Service Configuration API distribution is packaged in the file *serviceconfig-java-api-dist.tgz*, which is included in the SCA BB installation package.

- [Distribution Content, page 2-2](#)
- [Installing the Package, page 2-2](#)

## Distribution Content

The Service Configuration API distribution package installs the following folders and files:

- *<installation folder>*
  - *README*
- *<installation folder>\doc*
  - *serviceconfig-javadoc.zip*—The Service Configuration API Javadoc documentation
- *<installation folder>\lib*
  - *serviceconfigapi.jar*—The Service Configuration API library
  - *jdmkrt.jar*
- This folder may contain additional library JAR files that are necessary for the Service Configuration API operation.

## Installing the Package

To install the package, unpack the TGZ file (keeping the folder structure) to an empty folder.

- On Win32 platforms, use any common Windows compression utility to extract the file.
- On Linux/Solaris platforms, use:

```
#>tar -xvfpz serviceconfig-java-api-dist.tgz
```

The Service Configuration API is now installed and ready for use.

## Compiling and Running the Service Configuration API

In order to compile and run a program that uses the Service Configuration API, you must have *serviceconfigapi.jar* in the CLASSPATH.

For example, if your program source is in *MyApiClass.java*, your compilation command line should be:

```
#>javac -classpath .;<installation folder>\lib\serviceconfigapi.jar MyApiClass.java
```

To run the program, the command line should be:

```
#>java -cp .;<installation folder>\lib\serviceconfigapi.jar MyApiClass
```

## Running the Service Configuration API with Java 1.5

To use the Service Configuration API with Java 1.5, some library classes must precede the JRE classes in the boot classpath. Add the following argument to the command line:

```
-Xbootclasspath/p:<installation folder>/lib/jdmkrt.jar
```

For example:

```
#>java -Xbootclasspath/p:<installation folder>/lib/jdmkrt.jar -cp .;<installation folder>/lib/serviceconfigapi.jar MyApiClass
```

# Service Control Engine Platform Setup

The following sections describe the configuration that is performed on the Service Control Engine (SCE) platform to allow correct Service Configuration API functioning.

- [SCA BB Setup, page 2-3](#)
- [PRPC Server, page 2-3](#)

## SCA BB Setup

The Service Configuration API configures SCA BB, which should be installed on the SCE platform. For more information, see the *Cisco Service Control Application for Broadband User Guide*.

## PRPC Server

The Service Configuration API uses the Proprietary Remote Procedure Call (PRPC) protocol as a transport for the connection to the SCE. PRPC is a proprietary RPC protocol designed by Cisco. For more information, see the *Cisco SCMS SCE Subscriber API Programmer's Guide*.

Before using the Service Configuration API, ensure that:

- The SCE is up and running, and reachable from the machine that hosts the Service Configuration API
- The PRPC server on the SCE has started





# CHAPTER 3

## Programming with the SCA BB Service Configuration API

---

Revised: July 23, 2009, OL-7207-05

### Introduction

This chapter is a reference for the main classes and methods of the Cisco SCA BB Service Configuration API. It also contains programming guidelines and code examples.

- [Service Configuration API Packages, page 3-1](#)
- [Package `com.cisco.scabb.servconf.mgmt`, page 3-2](#)
- [Class `SCABB`, page 3-2](#)
- [Class `ConnectionApi`, page 3-5](#)
- [Class `ImportExportApi`, page 3-5](#)
- [Class `ServiceConfigApi`, page 3-16](#)
- [Class `ServiceConfig`, page 3-22](#)
- [Service Configuration API Programming Guidelines, page 3-30](#)
- [Service Configuration API Code Examples, page 3-31](#)

### Service Configuration API Packages

The Service Configuration API includes the following packages.

- Service Configuration Management API
  - `com.cisco.scabb.servconf.mgmt`
  - `com.pcube.apps.engage`
- Service Configuration Editing API
  - `com.cisco.scasbb.backend.classification`—Provides representation of the various model objects
  - `com.pcube.apps.engage.common`—Provides the classes that are used by the Policy and Subscriber classes
  - `com.pcube.apps.engage.policy`—Provides the classes that define service configurations

Only *com.cisco.scabb.servconf.mgmt* is documented in this guide. For details of the other packages, refer to the Javadoc that is part of the Service Configuration API distribution (see [Distribution Content](#), page 2-2).

## Package com.cisco.scabb.servconf.mgmt

Package *com.cisco.scabb.servconf.mgmt* contains the following classes:

- **SCABB**—Provides methods for connecting to the SCE platform; these methods are used for apply and retrieve operations.
- **ConnectionApi**—Provides the connection handle to the SCE platform.
- **ImportExportApi**—Provides methods for saving service configurations to and reading service configurations from PQB files, and for importing and exporting parts of a service configuration to and from CSV files.
- **ServiceConfigApi**—Provides methods for creating a new service configuration, and for applying service configurations to and retrieving service configurations from SCE platforms.
- **ServiceConfig**—Instances of this class are containers of configuration parameters that, when applied to the SCE platform, determine how the service control application performs classification, accounting and reporting, and control over network traffic.

## Class SCABB

Class SCABB supplies login and logout methods to the SCE. The handle returned by the login serves all of the Service Configuration API and Subscriber API methods that directly affect the SCE.

- [Class SCABB Methods](#), page 3-2

## Class SCABB Methods

Class SCABB methods are explained in the following sections.

- [login](#), page 3-3
- [login](#), page 3-3
- [logout](#), page 3-4
- [addNotificationListener](#), page 3-4
- [removeNotificationListener](#), page 3-4
- [getDefaultProtocolFamilies](#), page 3-5

## login

### Syntax

```
public static ConnectionApi login(String hostName,
                                String userName,
                                String password,
                                byte deviceType)
throws ConnectionFailedException
```

### Description

Connects to the SCE platform; the method returns a handle that is used by all the API methods that affect the device.

Every login operation must be closed with a *logout(ConnectionApi)* operation.

### Parameters

- **hostName**—The SCE host address.
- **userName**—The user name.
- **password**—The password.
- **deviceType**—The device type. Use device type *Connection.SE\_DEVICE* to connect to an SCE platform.

### Return Value

The connection, which is a handle passed to all the API methods.

### Exceptions

The following exception may be thrown by this method:

- **ConnectionFailedException**—Thrown if the login fails; you can retrieve the reason for the failure from the exception

## login

### Syntax

```
public static ConnectionApi login(com.pcube.management.framework.client.SessionObject
sessionObject)
```

### Description

Connects to a device by using an existing *SessionObject*; the method returns a handle that is used by all the API methods which affect the device.

Every login operation must be closed with a *logout(ConnectionApi)* operation.

### Parameters

- **sessionObject**—An existing *SessionObject* of the device

### Return Value

The connection, which is a handle passed to all the API methods.

## logout

### Syntax

```
public static void logout(ConnectionApi connectionApi)
```

### Description

Disconnects from the SCE; the connection cannot be used after this method is called.

### Parameters

- **connection**—The connection that keeps a handle to the connection to the SCE

## addNotificationListener

### Syntax

```
public static void addNotificationListener(NotificationListener listener)
```

### Description

Subscribes the specified SCABB notification listener to receive notifications regarding all operations performed by the SCABB API.

The listener must implement the NotificationListener interface.

### Parameters

- **listener**—The NotificationListener

## removeNotificationListener

### Syntax

```
public static void removeNotificationListener(NotificationListener listener)
```

### Description

Unsubscribes the SCABB listener from receiving notifications regarding SCABB API operations.

The removed listener will not receive any more notifications.

### Parameters

- **listener**—The NotificationListener to be removed

### Exceptions

The following exception may be thrown by this method:

- **IllegalArgumentException**—If the listener was never subscribed to listen to SCABB notifications

## getDefaultProtocolFamilies

**Syntax**

```
public static InputStream getDefaultProtocolFamilies()
```

**Description**

Gets the default protocol families for this class

**Return Value**

An InputStream of the protocol families

**See Also**

*Class.getResourceAsStream(java.lang.String)*

# Class ConnectionApi

Class ConnectionApi is a connection handle to the SCE that is used by all of the Service Configuration API and Subscriber API methods.

## Class ConnectionApi Methods

Class ConnectionApi contains one method, which is explained in the following section.

### isConnected

**Syntax**

```
public boolean isConnected()
```

**Description**

Checks if the connection is valid.

**Return Value**

`true` if connected to the device, `false` otherwise

# Class ImportExportApi

Class ImportExportApi is an API for import and export operations. Service configuration elements are imported and exported in CSV formats. Service configuration are imported and exported in XML format.

- [Class ImportExportApi Constructor, page 3-6](#)
- [Class ImportExportApi Methods, page 3-6](#)

## Class ImportExportApi Constructor

The Class ImportExportApi constructor is explained in the following section.

### ImportExportApi

#### Syntax

```
public ImportExportApi()
```

#### Description

The ImportExportApi constructor.

## Class ImportExportApi Methods

Class ImportExportApi methods are explained in the following sections.

- [exportServiceConfiguration, page 3-7](#)
- [importServiceConfiguration, page 3-7](#)
- [importFlavors, page 3-8](#)
- [importFlavors, page 3-8](#)
- [importZones, page 3-9](#)
- [importZones, page 3-9](#)
- [importProtocols, page 3-10](#)
- [importProtocols, page 3-10](#)
- [importServices, page 3-11](#)
- [importServices, page 3-11](#)
- [exportProtocols, page 3-12](#)
- [exportProtocols, page 3-12](#)
- [exportZones, page 3-13](#)
- [exportZones, page 3-13](#)
- [exportFlavors, page 3-14](#)
- [exportFlavors, page 3-14](#)
- [exportServices, page 3-15](#)
- [exportServices, page 3-15](#)
- [loadListArray, page 3-16](#)

## exportServiceConfiguration

### Syntax

```
public static void exportServiceConfiguration(ServiceConfig servConf,  
                                             File f)  
throws FileNotFoundException,  
     ImportErrorException
```

### Description

Exports a service configuration to a file.

### Parameters

- **servConf**—The service configuration to export
- **f**—The file to which to export the service configuration

### Exceptions

The following exceptions may be thrown by this method:

- `FileNotFoundException`
- `ImportExportException`—If an error occurs during export

## importServiceConfiguration

### Syntax

```
public static ServiceConfig importServiceConfiguration(File f)  
throws ImportErrorException,  
     IOException
```

### Description

Imports a service configuration from the specified file.

### Parameters

- **f**—The file containing the service configuration to import

### Return Value

The imported service configuration

### Exceptions

The following exceptions may be thrown by this method:

- `ImportExportException`—If an error occurs during import
- `IOException`

## importFlavors

### Syntax

```
public static void importFlavors(ServiceConfig servConf,
                                FlavorType flavorType,
                                File file)
throws ImportErrorException
```

### Description

Imports flavors of a specified type from a specified CSV file

### Parameters

- **servConf**—The service configuration into which to import the flavors
- **flavorType**—The type of the imported flavors
- **file**—The CSV file from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during import

## importFlavors

### Syntax

```
public static void importFlavors(ServiceConfig servConf,
                                FlavorType flavorType,
                                InputStream inStream)
throws ImportErrorException
```

### Description

Imports flavors of a specified type from a given input-stream

### Parameters

- **servConf**—The service configuration into which to import the flavors
- **flavorType**—The type of the imported flavors
- **inStream**—The input stream from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during import

## importZones

### Syntax

```
public static void importZones(ServiceConfig servConf,  
                             File file)  
throws ImportExportException
```

### Description

Imports zones from a specified CSV file

### Parameters

- **servConf**—The service configuration into which to import the zones
- **file**—The CSV file from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during import

## importZones

### Syntax

```
public static void importZones(ServiceConfig servConf,  
                             InputStream inStream)  
throws ImportExportException
```

### Description

Imports zones from a given input stream

### Parameters

- **servConf**—The service configuration into which to import the zones
- **inStream**—The input stream from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during import

## importProtocols

### Syntax

```
public static void importProtocols(ServiceConfig servConf,
                                  File file)
throws ImportErrorException
```

### Description

Imports protocols from a specified CSV file

### Parameters

- **servConf**—The service configuration into which to import the protocols
- **file**—The CSV file from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportErrorException**—If an error occurs during import

## importProtocols

### Syntax

```
public static void importProtocols(ServiceConfig servConf,
                                  InputStream inStream)
throws ImportErrorException
```

### Description

Imports protocols from a given input stream

### Parameters

- **servConf**—The service configuration into which to import the protocols
- **inStream**—The input stream from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportErrorException**—If an error occurs during import

## importServices

### Syntax

```
public static void importServices(ServiceConfig servConf,  
                                File file)  
throws ImportExportException
```

### Description

Imports services from a specified CSV file

### Parameters

- **servConf**—The service configuration into which to import the services
- **file**—The CSV file from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during import

## importServices

### Syntax

```
public static void importServices(ServiceConfig servConf,  
                                InputStream inStream)  
throws ImportExportException
```

### Description

Imports services from a given input stream

### Parameters

- **servConf**—The service configuration into which to import the services
- **inStream**—The input stream from which to import

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during import

## exportProtocols

### Syntax

```
public static void exportProtocols(List protocols,
                                   File file)
throws ImportErrorException
```

### Description

Exports protocols to a specified file in a CSV format

### Parameters

- **protocols**—The list of protocols to export
- **file**—The file to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## exportProtocols

### Syntax

```
public static void exportProtocols(List protocols,
                                   OutputStream outputStream)
throws ImportErrorException
```

### Description

Exports protocols to a given output stream in a CSV format

### Parameters

- **protocols**—The list of protocols to export
- **outStream**—The output stream to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## exportZones

### Syntax

```
public static void exportZones(List zones,
                               File file)
throws ImportErrorException
```

### Description

Exports zones to a specified file in a CSV format

### Parameters

- **zones**—The list of zones to export
- **file**—The file to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## exportZones

### Syntax

```
public static void exportZones(List zones,
                               OutputStream outputStream)
throws ImportErrorException
```

### Description

Exports zones to a given output stream in a CSV format

### Parameters

- **zones**—The list of zones to export
- **outStream**—The output stream to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## exportFlavors

### Syntax

```
public static void exportFlavors(List flavors,
                                FlavorType flavorType,
                                File file)

throws ImportErrorException
```

### Description

Exports flavors of a specified type to a specified file in a CSV format

### Parameters

- **flavors**—The list of flavors to export
- **flavorType**—The type of the exported flavors
- **file**—The file to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## exportFlavors

### Syntax

```
public static void exportFlavors(List flavors,
                                FlavorType flavorType,
                                OutputStream outputStream)

throws ImportErrorException
```

### Description

Exports flavors of a specified type to a given output stream in a CSV format

### Parameters

- **flavors**—The list of flavors to export
- **flavorType**—The type of the exported flavors
- **outStream**—The output stream to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## exportServices

### Syntax

```
public static void exportServices(ServiceConfig servConf,  
                                File file)  
throws ImportErrorException
```

### Description

Exports services from a service configuration to a file in CSV format

### Parameters

- **servConf**—The service configuration from which to export the services
- **file**—The file to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## exportServices

### Syntax

```
public static void exportServices(ServiceConfig servConf,  
                                OutputStream outputStream)  
throws ImportErrorException
```

### Description

Exports services from a service configuration to an output stream in CSV format

### Parameters

- **servConf**—The service configuration from which to export the services
- **outStream**—The output stream to which to export

### Exceptions

The following exception may be thrown by this method:

- **ImportExportException**—If an error occurs during export

## loadListArray

### Syntax

```
public static void loadListArray(ServiceConfig servConf,
                               InputStream inStream)
    throws ImportExportException
```

### Description

Deprecated—Only used for 2.57 host list and IP list:

- It converts 2.57 CSV host-list files into 3.0 CSV HTTP URL flavor files
- It converts 2.57 CSV IP-list files into 3.0 CSV zone files

For loading 3.0 CSV files use the methods `importFlavors(ServiceConfig, FlavorType, File)` and `importZones(ServiceConfig, File)`

### Parameters

- **servConf**—The service configuration into which to import the services
- **inStream**—The input stream to import to

### Exceptions

The following exception may be thrown by this method:

- `ImportExportException`

## Class ServiceConfigApi

Class `ServiceConfigApi` exposes Service Configuration API methods. All of these methods get or set data located in the SCE, and therefore require a connection to the SCE platform.

## Class ServiceConfigApi Methods

Class `ServiceConfigApi` methods are explained in the following sections.

- [applyServiceConfiguration, page 3-17](#)
- [applyServiceConfiguration, page 3-17](#)
- [applyServiceConfiguration, page 3-18](#)
- [retrieveServiceConfiguration, page 3-19](#)
- [updateValuesIni, page 3-19](#)
- [updateValuesIni, page 3-20](#)
- [validateServiceConfiguration, page 3-20](#)
- [importServConf, page 3-21](#)
- [exportServConf, page 3-21](#)
- [importDefaultServConf, page 3-22](#)

## applyServiceConfiguration

### Syntax

```
public static long applyServiceConfiguration(ConnectionApi connectionApi,  
                                           ServiceConfig servConf)  
  
throws ElementManagementException,  
       ApplyException
```

### Description

Applies a service configuration to the specified SCE.

### Parameters

- **connectionApi**—The ConnectionApi, which keeps a handle to the connection to the SCE
- **servConf**—The service configuration to apply

### Return Value

The operation timestamp

### Exceptions

The following exceptions may be thrown by this method:

- ElementManagementException
- ApplyException

## applyServiceConfiguration

### Syntax

```
public static long applyServiceConfiguration(ConnectionApi connectionApi,  
                                           ServiceConfig servConf,  
                                           Properties applySettings)  
  
throws ElementManagementException,  
       ApplyException
```

### Description

Applies a service configuration to the specified SCE.

### Parameters

- **connectionApi**—The ConnectionApi, which keeps a handle to the connection to the SCE
- **servConf**—The service configuration to apply
- **applySettings**—Properties

### Return Value

The operation timestamp

### Exceptions

The following exceptions may be thrown by this method:

- ElementManagementException
- ApplyException

## applyServiceConfiguration

### Syntax

```
public static long applyServiceConfiguration(ConnectionApi connectionApi,
                                           ServiceConfig servConf,
                                           boolean updateCm,
                                           Properties cmIpRemap,
                                           int cmUpdateMethod,
                                           int rpcPort
                                           boolean forceTemplateVirtualLinksInVlMode)
throws ElementManagementException,
       ApplyException
```

### Description

Applies a service configuration to the specified SCE.

### Parameters

- **connectionApi**—The ConnectionApi, which keeps a handle to the connection to the SCE
- **servConf**—The service configuration to apply
- **updateCm**—Whether the Collection Manager for the specified SCE will be updated with the specified service configuration values
- **cmIpRemap**—A map from the CM IP address as configured in the SCE, to the actual CM addresses
- **cmUpdateMethod**—The method to use to connect to the CM
- **rpcPort**—Port number for the CM RPC connection
- **forceTemplateVirtualLinksInVlMode**—

### Return Value

The operation timestamp

### Exceptions

The following exceptions may be thrown by this method:

- ElementManagementException
- ApplyException

### See Also

*PolicyAPI.DC\_UPDATE\_METHOD\_RPC* , *PolicyAPI.DC\_DEFAULT\_RPC\_PORT*

## retrieveServiceConfiguration

### Syntax

```
public static ServiceConfig retrieveServiceConfiguration(ConnectionApi connectionApi)
throws IOException,
      ElementManagementException,
      ApplyException
```

### Description

Retrieves the service configuration loaded in the specified SCE.

### Parameters

- **connectionApi**—The ConnectionApi that keeps a handle to the connection to the SCE

### Return Value

The service configuration in the SCE platform

### Exceptions

The following exceptions may be thrown by this method:

- IOException
- ElementManagementException
- ApplyException

## updateValuesIni

### Syntax

```
public static void updateValuesIni(String cmAddress,
                                  String sceAddress,
                                  ServiceConfig servConf)
throws ApplyException
```

### Description

Updates the CM with data from the SCE platform's service configuration.

### Parameters

- **cmAddress**—The address of the CM to update
- **sceAddress**—The address of the SCE platform that enforces the given service configuration
- **servConf**—The service configuration

### Exceptions

The following exception may be thrown by this method:

- ApplyException

## updateValuesIni

### Syntax

```
public static void updateValuesIni(String cmAddress,
                                  String sceAddress,
                                  ServiceConfig servConf,
                                  int cmUpdateMethod,
                                  int rpcPort)
```

throws ApplyException

### Description

Updates the CM with data from the SCE platform's service configuration.

### Parameters

- **cmAddress**—The address of the CM to update
- **sceAddress**—The address of the SCE platform that enforces the given service configuration
- **servConf**—The service configuration
- **cmUpdateMethod**
- **rpcPort**

### Exceptions

The following exception may be thrown by this method:

- ApplyException

## validateServiceConfiguration

### Syntax

```
public static ArrayList validateServiceConfiguration(ServiceConfig servConf)
```

### Description

Validates a service configuration

### Parameters

- **servConf**—The service configuration to validate

### Return Value

A vector with warning messages regarding rules that might have undesirable results

### See Also

*PolicyValidator.validatePolicy(com.pcube.apps.engage.policy.Policy)*

## importServConf

### Syntax

```
public static ServiceConfig importServConf(File pqbfile)
throws ImportExportException,
      IOException
```

### Description

Loads the service configuration from a PQB file.

### Parameters

- **pqbfile**—The PQB file to load

### Return Value

The resulting service configuration

### Exceptions

The following exceptions may be thrown by this method:

- ImportExportException
- IOException

## exportServConf

### Syntax

```
public static void exportServConf(ServiceConfig servConf,
                                  File pqbfile)
throws FileNotFoundException,
      ImportExportException
```

### Description

Saves a service configuration to a PQB file

### Parameters

- **servConf**—The service configuration to save
- **pqbfile**—The PQB file into which to save the service configuration

### Exceptions

The following exceptions may be thrown by this method:

- FileNotFoundException
- ImportExportException

## importDefaultServConf

### Syntax

```
public static ServiceConfig importDefaultServConf()
throws ImportExportException
```

### Description

Loads the default service configuration

### Return Value

The default service configuration

### Exceptions

The following exception may be thrown by this method:

- `ImportExportException`

## Class ServiceConfig

Class `ServiceConfig` is the whole set of lists, protocols, services, and packages that an ISP has defined. The service configuration is applied to the `ServiceConfig Domain`'s SCE platforms and sets their application parameters to regulate subscribers' flows.

## Class ServiceConfig Methods

Class `ServiceConfig` methods are explained in the following sections.

- [getCalendarList](#), page 3-23
- [getClassificationCfg](#), page 3-23
- [getDynamicSignatureScript](#), page 3-23
- [getPackageList](#), page 3-24
- [getPolicySettings](#), page 3-24
- [getProtocolList](#), page 3-24
- [getProtocolRedirectIndexNameArray](#), page 3-24
- [getProtocolRedirectString](#), page 3-25
- [getRealTimeFrameName](#), page 3-25
- [getServiceList](#), page 3-26
- [getSubNotifications](#), page 3-26
- [getTimeFrameNames](#), page 3-26
- [getProtocolRedirectString](#), page 3-27
- [getZoneList](#), page 3-27
- [isProtocolRedirectable](#), page 3-28
- [setProtocolRedirectString](#), page 3-28

- [setProtocolRedirectString](#), page 3-29
- [setTimeFrameName](#), page 3-29
- [setTimeFrameName](#), page 3-30
- [setTimeFrameNames](#), page 3-30

## getCalendarList

### Syntax

```
public CalendarArray getCalendarList()
```

### Description

Gets the calendar list

### Return Value

The list of calendars in the service configuration

## getClassificationCfg

### Syntax

```
public ClassificationConfiguration getClassificationCfg()
```

### Description

Gets the classification configuration

### Return Value

The classification-related configuration in the domain

## getDynamicSignatureScript

### Syntax

```
public DynamicSignaturesScript getDynamicSignatureScript()
```

### Description

Gets the dynamic-signature configuration

### Return Value

The dynamic-signature script

## getPackageList

**Syntax**

```
public PackageArray getPackageList()
```

**Description**

Gets this service configuration's package list.

**Return Value**

This service configuration's package list

## getPolicySettings

**Syntax**

```
public PolicySettings getPolicySettings()
```

**Description**

Gets this service configuration's settings. These settings are general system settings for the SCE platforms in this service configuration's domain.

**Return Value**

This service configuration's settings

## getProtocolList

**Syntax**

```
public ProtocolList getProtocolList()
```

**Description**

Gets this service configuration's protocol list. The protocols in this list are referred to by the service configuration's services.

**Return Value**

This service configuration's protocols list

## getProtocolRedirectIndexNameArray

**Syntax**

```
public ProtocolRedirectIndexNameArray getProtocolRedirectIndexNameArray()
```

**Description**

Gets the list of the protocols' redirect index names.

**Return Value**

The list of the protocols' redirect index names

## getProtocolRedirectString

### Syntax

```
public String getProtocolRedirectString(String protocolName,  
                                     int redirectIndex)  
throws ItemNotFoundException
```

### Description

Gets the redirect address for the protocol in a certain index in its redirect String array.

Redirection is part of the protocol's specification, and exists for only a few predefined protocols.

### Parameters

- **protocolName**—The queried protocol
- **redirectIndex**—The index in the redirect String array

### Return Value

The redirect address

### Exceptions

The following exception may be thrown by this method:

- **ItemNotFoundException**—If there is no such predefined protocol in this service configuration's ProtocolArray, or if the redirect index is out of bound

## getRealTimeFrameName

### Syntax

```
public String getRealTimeFrameName(String name)  
throws ItemNotFoundException
```

### Description

Gets the predefined API name for a certain TimeFrame.

### Parameters

- **name**—This service configuration's alias for the TimeFrame

### Return Value

The TimeFrame's API name

### Exceptions

The following exception may be thrown by this method:

- **ItemNotFoundException**—If there is no such alias in this service configuration

## getServiceList

**Syntax**

```
public ServiceArray getServiceList()
```

**Description**

Gets this service configuration's service list.

**Return Value**

This service configuration's service list

## getSubNotifications

**Syntax**

```
public SubNotificationArray getSubNotifications()
```

**Description**

Gets the subscriber notifications as configured in the service configuration

**Return Value**

The subscriber notifications

## getTimeFrameNames

**Syntax**

```
public String[] getTimeFrameNames()
```

**Description**

Gets the time-frame names this service configuration has assigned to the different TimeFrames.

These aliases allow time frames to have meaningful names.

The returned String array keeps in index X the alias of the TimeFrame index X.

**Return Value**

The String array of this service configuration's time-frame names

## getProtocolRedirectString

### Syntax

```
public String getProtocolRedirectString(String protocolName)
throws ItemNotFoundException
```

### Description

Gets the default redirect address for the protocol.

Redirection is part of the protocol's specification, and exists for only a few predefined protocols.

### Parameters

- **protocolName**—The queried protocol

### Return Value

The redirect address

### Exceptions

The following exception may be thrown by this method:

- **ItemNotFoundException**—If there is no such predefined protocol in this service configuration's ProtocolArray

## getZoneList

### Syntax

```
public ZoneList getZoneList()
```

### Description

Gets this service configuration's IP, IP ranges, and host lists array. These lists are referred to by this service configuration's services.

### Return Value

This service configuration's array of lists

## isProtocolRedirectable

### Syntax

```
public boolean isProtocolRedirectable(String protocolName)
throws ItemNotFoundException
```

### Description

Checks if a specified protocol supports redirecting.

Redirection is part of the protocol's specification, and exists for only a few predefined protocols.

### Parameters

- **protocolName**—The queried protocol's name

### Return Value

`true` if the protocol support redirection, `false` otherwise

### Exceptions

The following exception may be thrown by this method:

- `ItemNotFoundException`—If there is no such predefined protocol in this service configuration's `ProtocolArray`

## setProtocolRedirectString

### Syntax

```
public void setProtocolRedirectString(String protocolName,
                                     int redirectIndex,
                                     String value)
throws ItemNotFoundException,
       MalformedURLException
```

### Description

Sets the address to which to redirect a certain flow. The redirection occurs when a rule has an `ACCESS_BLOCK_AND_REDIRECT` access mode for a certain service that uses the desired protocol. The setting is done to a certain String in the redirect String array. The rule chooses which redirect String to use from the redirect String array.

### Parameters

- **protocolName**—The protocol whose activity will be redirected
- **redirectIndex**—The redirect String index in the protocol's redirect String array
- **value**—The protocol's redirect address

### Exceptions

The following exceptions may be thrown by this method:

- `ItemNotFoundException`—If there is no such predefined protocol in this service configuration's `ProtocolArray`, or if the redirect index is out of bound
- `MalformedURLException`—If the String is an illegal URL name

## setProtocolRedirectString

### Syntax

```
public void setProtocolRedirectString(String protocolName,  
                                     String value)  
  
throws ItemNotFoundException,  
       MalformedURLException
```

### Description

Sets the default address to which to redirect a certain flow. The redirection occurs when a rule has an *ACCESS\_BLOCK\_AND\_REDIRECT* access mode for a certain service that uses the desired protocol.

### Parameters

- **protocolName**—The desired protocol whose activity will be redirected
- **value**—The protocol's redirect address

### Exceptions

The following exceptions may be thrown by this method:

- *ItemNotFoundException*—If there is no such predefined protocol in this service configuration's *ProtocolArray*
- *MalformedURLException*—If the *String* is an illegal URL name

## setTimeFrameName

### Syntax

```
public void setTimeFrameName(int index,  
                             String newName)  
  
throws ItemNotFoundException,  
       DuplicateItemException
```

### Description

Sets an alias for the *TimeFrame* that has a given index.

The alias allows the time frame to have a meaningful name.

### Parameters

- **index**—The index of the *TimeFrame* to which the alias is given
- **newName**—The alias

### Exceptions

The following exceptions may be thrown by this method:

- *ItemNotFoundException*—If there is no such *TimeFrame*
- *DuplicateItemException*

## setTimeFrameName

### Syntax

```
public void setTimeFrameName(TimeFrame frame,
                             String newName)
throws ItemNotFoundException,
       DuplicateItemException
```

### Description

Sets an alias for the specified TimeFrame.

The alias allows the time frame to have a meaningful name.

### Parameters

- `frame`—The TimeFrame to which the alias is given
- `newName`—The alias

### Exceptions

The following exceptions may be thrown by this method:

- `ItemNotFoundException`—If there is no such TimeFrame
- `DuplicateItemException`

## setTimeFrameNames

### Syntax

```
public void setTimeFrameNames(String[] newNames)
```

### Description

Sets the names of all time frames.

### Parameters

- `newNames`—The names to set

# Service Configuration API Programming Guidelines

## Connections to the SCE Platform

Make sure that connections you create using any of the `login()` methods are properly closed using `logout()` (see [logout](#), page 3-4).

# Service Configuration API Code Examples

This section gives several code examples of Service Configuration API usage.

- [Applying a Service Configuration, page 3-31](#)
- [Updating Zones Automatically, page 3-33](#)
- [Listing Names of Services and Packages, page 3-36](#)

## Applying a Service Configuration

The following example applies a new service configuration to the SCE platform that is specified in the command line.

```
package examples;

import java.io.File;

import com.cisco.scabb.servconf.mgmt.ConnectionApi;
import com.cisco.scabb.servconf.mgmt.ImportExportApi;
import com.cisco.scabb.servconf.mgmt.SCABB;
import com.cisco.scabb.servconf.mgmt.ServiceConfig;
import com.cisco.scabb.servconf.mgmt.ServiceConfigApi;
import com.pcube.apps.engage.Connection;
import com.pcube.apps.engage.ConnectionFailedException;

/**
 * applies the service configuration in the PQB file to the SCE
 * specified in the command line. message is printed to standard error
 * in case of failure.
 * <p>
 * usage: java examples.SimpleApplyPqb <sce-address> <password>
 * <pqb-filename>
 */
public class SimpleApplyPqb {

    public static void main(String[] args) {

        if (args.length != 3) {
            System.err.println("usage: java examples.SimpleApplyPqb "
                + "<sce-address> <password> <pqb-filename>");
            System.exit(1);
        }

        String sceAddress = args[0];
        String password = args[1];
        String pqbFilename = args[2];

        ServiceConfig serviceConfig = openPqbFile(pqbFilename);

        if (serviceConfig == null) {
            return;
        }

        applyPqb(sceAddress, password, serviceConfig);
    }

    /**
     * apply the service configuration in the specified PQB file to the
```

```

* specified SCE. message is printed to standard error in case of
* failure.
*
* @param sceAddress
* @param password
* @param serviceConfig
*/
private static void applyPqb(String sceAddress, String password,
    ServiceConfig serviceConfig) {
    ConnectionApi connection = null;
    try {

        System.out.println("connecting to SCE at " + sceAddress);
        connection = SCABB.login(sceAddress, "admin", password,
            Connection.SE_DEVICE);
        System.out.println("connected to SCE");

        System.out.println("applying service configuration");
        ServiceConfigApi.applyServiceConfiguration(connection,
            serviceConfig);
        System.out.println("service configuration applied");

    } catch (ConnectionFailedException e) {

        System.err.println("connection to SCE failed: "
            + e.getMessage());
        e.printStackTrace();

    } catch (Exception e) {

        System.err.println("apply operation failed: "
            + e.getMessage());
        e.printStackTrace();

    } finally {

        if (connection != null) {
            System.out.println("disconnecting from SCE");
            SCABB.logout(connection);
            System.out.println("disconnected");
        }

    }
}

/**
* return the service configuration in the specified PQB file, or
* null if reading the file has failed. message is printed to
* standard error in case of failure.
*
* @param pqbFilename
* @return
*/
private static ServiceConfig openPqbFile(String pqbFilename) {

    ServiceConfig serviceConfig = null;
    try {

        System.out.println("opening PQB file " + pqbFilename);
        serviceConfig = ImportExportApi
            .importServiceConfiguration(new File(pqbFilename));
        System.out.println("PQB file opened");

    } catch (Exception e) {

```

```

        System.err.println("opening PQB file failed: "
            + e.getMessage());
        e.printStackTrace();
    }
    return serviceConfig;
}
}

```

## Updating Zones Automatically

The following example updates an SCE with zone IP addresses. The zone IP addresses are specified in a CSV file.

```

package examples;

import java.io.File;

import com.cisco.scabb.servconf.mgmt.ConnectionApi;
import com.cisco.scabb.servconf.mgmt.ImportExportApi;
import com.cisco.scabb.servconf.mgmt.SCABB;
import com.cisco.scabb.servconf.mgmt.ServiceConfig;
import com.cisco.scabb.servconf.mgmt.ServiceConfigApi;
import com.cisco.scasbb.backend.classification.Zone;
import com.pcube.apps.engage.Connection;
import com.pcube.apps.engage.ConnectionFailedException;
import com.pcube.apps.engage.common.ImportExportException;

/**
 * updates an SCE with zone IP addresses. the zone IP address are
 * specified in a CSV file. the SCE address and CSV filename are taken
 * from the cmd-line argument.
 * <p>
 * usage: java examples.UpdateZoneFromCsv <sce-address> <password>
 * <zone-csv-file> <zone-name>
 *
 */
public class UpdateZoneFromCsv {

    public static void main(String[] args) {

        if (args.length != 4) {
            System.err.println("usage: java examples.UpdateZoneFromCsv"
                + " <sce-address> <password>"
                + " <zone-csv-file> <zone-name>");
            System.exit(1);
        }

        String sceAddress = args[0];
        String password = args[1];
        String csvFilename = args[2];
        String zoneName = args[3];

        ServiceConfig serviceConfig = retrievePqb(sceAddress, password);
        if (serviceConfig == null) {
            return;
        }

        ServiceConfig updatedServiceConfig = importZoneFromCsv(
            serviceConfig, csvFilename, zoneName);
    }
}

```

```

    if (updatedServiceConfig == null) {
        return;
    }

    applyPqb(sceAddress, password, updatedServiceConfig);
}

/**
 * apply the service configuration in the specified PQB file to the
 * specified SCE. message is printed to standard error in case of
 * failure.
 *
 * @param sceAddress
 * @param password
 * @param serviceConfig
 */
private static void applyPqb(String sceAddress, String password,
    ServiceConfig serviceConfig) {

    ConnectionApi connection = null;
    try {

        System.out.println("connecting to SCE at " + sceAddress);
        connection = SCABB.login(sceAddress, "admin", password,
            Connection.SE_DEVICE);
        System.out.println("connected to SCE");

        System.out.println("applying service configuration");
        ServiceConfigApi.applyServiceConfiguration(connection,
            serviceConfig);
        System.out.println("service configuration applied");

    } catch (ConnectionFailedException e) {

        System.err.println("connection to SCE failed: "
            + e.getMessage());
        e.printStackTrace();

    } catch (Exception e) {

        System.err.println("apply operation failed: "
            + e.getMessage());
        e.printStackTrace();

    } finally {

        if (connection != null) {
            System.out.println("disconnecting from SCE");
            SCABB.logout(connection);
            System.out.println("disconnected");
        }

    }
}

private static ServiceConfig importZoneFromCsv(
    ServiceConfig serviceConfig, String csvFilename,
    String zoneName) {

    // clear zone items
    Zone zone = (Zone) serviceConfig.getClassificationCfg()
        .getZoneList().findByName(zoneName);
    if (zone == null) {

```

```
        System.err.println("WARNING: zone not found: " + zoneName);
    } else {

        zone.getZoneItems().clear();

    }

    // import new zone items
    try {

        ImportExportApi.importZones(serviceConfig, new File(
            csvFilename));

    } catch (ImportExportException e) {

        System.err.println("importing zones failed: "
            + e.getMessage());
        e.printStackTrace();
        return null;

    }

    return serviceConfig;
}

private static ServiceConfig retrievePqb(String sceAddress,
    String password) {

    ServiceConfig retrievedServiceConfig = null;
    ConnectionApi connection = null;
    try {

        System.out.println("connecting to SCE at " + sceAddress);
        connection = SCABB.login(sceAddress, "admin", password,
            Connection.SE_DEVICE);
        System.out.println("connected to SCE");

        System.out.println("retrieving service configuration");
        retrievedServiceConfig = ServiceConfigApi
            .retrieveServiceConfiguration(connection);
        System.out.println("service configuration retrieved");

    } catch (ConnectionFailedException e) {

        System.err.println("connection to SCE failed: "
            + e.getMessage());
        e.printStackTrace();

    } catch (Exception e) {

        System.err.println("retrieve operation failed: "
            + e.getMessage());
        e.printStackTrace();

    } finally {

        if (connection != null) {
            System.out.println("disconnecting from SCE");
            SCABB.logout(connection);
            System.out.println("disconnected");
        }
    }
}
```

```

    }
    return retrievedServiceConfig;
}
}

```

## Listing Names of Services and Packages

The following example prints the names of the services and packages that are in a service configuration.

```

package examples;

import java.io.File;
import java.io.IOException;
import java.util.Iterator;

import com.cisco.scabb.servconf.mgmt.ImportExportApi;
import com.cisco.scabb.servconf.mgmt.ServiceConfig;
import com.cisco.scabb.servconf.mgmt.ServiceConfigApi;
import com.pcube.apps.engage.common.ImportExportException;
import com.pcube.apps.engage.policy.Package;
import com.pcube.apps.engage.policy.Service;

public class IterateServiceConfig {

    public static void main(String[] args)
        throws ImportExportException, IOException {

        // take the PQB filename from the cmd-line, or use the default
        // service configuration instead
        ServiceConfig serviceConfig = null;
        if (args.length > 0) {
            serviceConfig = ImportExportApi
                .importServiceConfiguration(new File(args[0]));
        } else {
            serviceConfig = ServiceConfigApi.importDefaultServConf();
        }

        System.out.println("----- package names -----");
        Iterator pkgIter = serviceConfig.getPackageList().iterator();
        while (pkgIter.hasNext()) {

            Package pkg = (Package) pkgIter.next();
            System.out.println(pkg.getNumericId() + ": "
                + pkg.getName());

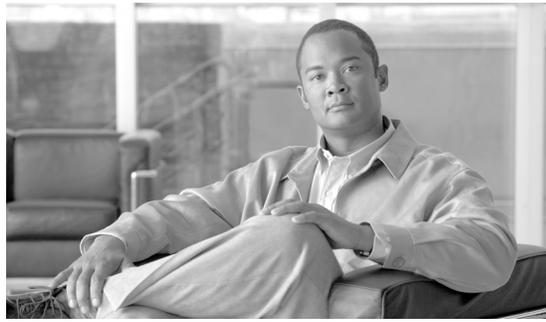
        }

        System.out.println("----- service names -----");
        Iterator svcIter = serviceConfig.getServiceList().iterator();
        while (svcIter.hasNext()) {

            Service svc = (Service) svcIter.next();
            System.out.println(svc.getNumericId() + ": "
                + svc.getName());

        }
    }
}

```



## CHAPTER 4

# Logging and Troubleshooting

---

Revised: July 23, 2009, OL-7207-05

## Introduction

This chapter describes the Cisco SCA BB Service Configuration API logging functionality. API logging allows you to monitor the operations being called by the API client.

API logging also allows you to troubleshoot the Service Configuration API integration.

## Service Configuration API Client Logging

The Service Configuration API uses the Apache Jakarta log4j package for logging. This provides the ability to log every activated operation into the *apilog* file located in the *\${user.home}* directory.

Logging parameters are configured using the log4J properties files. To enable logging, make sure that this file is in the application's CLASSPATH. The file is read at startup of the application. If you make any changes you must restart the application.

The following is the installed content of the *log4j.properties* file:

```
# default Log4j configuration for Service Configuration API
log4j.rootCategory=INFO, apiStdout
# In order to enable the logging to the file Replace the above
# line with the following:
# log4j.rootCategory=INFO, files
# stdout is set to be a ConsoleAppender.
log4j.appender.apiStdout=org.apache.log4j.ConsoleAppender
log4j.appender.apiStdout.layout=org.apache.log4j.PatternLayout
log4j.appender.apiStdout.layout.ConversionPattern=%d{dd-MMM HH:mm:ss.SSS} [%t] %-5p
%c%n%m%n
# files is set to be a RollingFileAppender.
#log4j.appender.files=org.apache.log4j.RollingFileAppender
#log4j.appender.files.layout=org.apache.log4j.PatternLayout
#log4j.appender.files.layout.ConversionPattern=%d{dd-MMM yyyy HH:mm:ss.SSS} [%t] %-5p %c
%x%n%m%n
#log4j.appender.files.File=${user.home}/apilog
#log4j.appender.files.Threshold=INFO
#log4j.appender.files.ImmediateFlush=true
#log4j.appender.files.MaxFileSize=1MB
#log4j.appender.files.MaxBackupIndex=4
# In order to enable debug logging uncomment the following two lines
#log4j.category.com.cisco=DEBUG
#log4j.category.com.pcube=DEBUG
```

To enable debug logging, uncomment the last two lines in the file. By default, logging is sent to the standard output.

To direct the logging to a file, uncomment the line:

```
# log4j.rootCategory=INFO, files
```

as explained in the file.