Cisco SCA BB Service Configuration API Programmer Guide

Release 4.0.x
May 27, 2013

Cisco Systems, Inc.
www.cisco.com

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Text Part Number: OL-29109-01

Cisco SCA BB Service Configuration API Programmer Guide

Preface

Revised: May 27, 2013, OL-29109-01

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 (SCA) running on a Service Control Engine (SCE) platform.

Document Revision History

Table 1 records changes to this document.

Table 1 Document Revision History

Revision	Cisco Service Control Release and Date	Change Summary
OL-26820-01	Supports all 4.0.x releases September 17, 2012	Updated for release 4.0.0.

Organization

Table 2 lists the major sections of this guide:

Table 2 Documentation Organization

Chapter	Title	Description
Chapter 1	SCA BB Service Configuration API Overview	Describes the various terms and concepts that are used when working with the Cisco Service Control Application for Broadband (SCA BB) Service Configuration API (Service Configuration API).
Chapter 2	Getting Started	Explains how to install, compile, and run the Service Configuration API.

Table 2 Documentation Organization (continued)

Chapter	Title	Description
Chapter 3	Programming with the SCA BB Service Configuration API	Provides details on the Service Configuration API programming structure and classes, and the methods of the main Service Configuration Management API class. This chapter also provides programming guidelines and code examples.
Chapter 4	Logging and Troubleshooting	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 SCA BB:

- *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

Table 3 lists the conventions used in this document.

Table 3 Documentation Conventions

Convention	Description
boldface font	Commands and keywords are in **boldface**.
italic font	Arguments for which you supply values are in *italics*.
[ ]	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 `screen` font.
`boldface screen` font	Information you must enter is in **`boldface screen`** font.

Table 3 Documentation Conventions (continued)

Convention	Description
italic screen font	Arguments for which you supply values are in *italic screen* font.
<>	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.

SCA BB Service Configuration API Overview

Revised: May 27, 2013, OL-29109-01

Introduction

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

This chapter consists of these sections:

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. These parameters together determine how the Cisco SCA, which runs on the Cisco 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. Cisco SCA BB is a GUI tool that lets you edit every aspect of the configuration and apply changes to the SCE platform.

The Service Configuration API enables external applications to change the policy that SCE platform enforces, 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:

- Service Configuration Management API—Handles the tasks that are performed on a service configuration as a whole. For example, 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. For example, changing a classification criterion or adding a policy rule.

This section contains these topics:

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 (created by the API and 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 comma-separated values (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 *Cisco Service Control Application for Broadband User Guide*. The CSV formats are described in the “CSV File Formats” chapter of *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—Provides the connection to the SCE platform.
- ImportExportApi—Provides methods for saving service configurations to and reading service configurations from PQB files. This API also provides methods 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—Determines how the SCA performs classification, accounting and reporting, and control over network. Objects of this class are basically containers of configuration parameters.

Service Configuration Editing API

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

- Inspecting a service configuration—Inspects a configuration that is retrieved from an SCE platform to determine, for example, which services and subscriber packages it contains. This API can be used for comparing two configurations.
- Configuring criteria for traffic classification—Determines how traffic is classified according to various criteria such as 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—Determines how network traffic is controlled by using rules for each service and for each subscriber package, elaborate bandwidth control hierarchy, and quota management models. The API can be used to change these settings, thus 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 by using this API. Use the Subscriber APIs described in the SM API programmer guides. See *Cisco SCMS SM Java API Programmer Guide* or the *Cisco SCMS SM C/C++ API Programmer Guide*.

Note Monitoring counters cannot be performed by using this API. You can monitor counters via SNMP to the SCE platform or via SQL to the Cisco Service Control Management Suite Collection Manager database. For details on monitoring via SNMP, see the "SCA BB Proprietary MIB Reference" chapter of *Cisco Service Control Application for Broadband Reference Guide.* For details on monitoring via SQL, see the *Cisco Service Control Management Suite Collection Manager User Guide*.

Service Configuration Editing API Data Types

The Service Configuration editing API data types for:

- Traffic classification settings are services, protocols, signatures, zones, and flavors.
- Traffic accounting and reporting settings are counter definitions and various Raw Data Records (RDR) parameters.
- Traffic control settings are packages, rules, subscriber bandwidth controllers, and global bandwidth controllers.

An explanation of the objects (service configuration entities) that make up a service configuration is beyond the scope of this document. See the "Traffic Processing Overview" chapter of *Cisco Service Control Application for Broadband User Guide*.

Use the SCA BB Console, which uses Service Configuration Editing API, to get familiar with these configuration objects and to understand the capabilities of 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.

   For code example, see the "Updating Zones Automatically" section on page 3-30.
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.

   For code example, see the "Applying a Service Configuration" section on page 3-28.

Another common workflow uses 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 response to some external trigger or at specified times.

Getting Started

Revised: May 27, 2013, OL-29109-01

Introduction

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

System Requirements

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

Compiling and running Java programs by using the Service Configuration API requires Java Development Kit (JDK) and Java Runtime Environment (JRE) versions 1.4 or 1.5.

Note To use the API with Java Version 1.5, you must set a special JRE option (see the "Running the Service Configuration API with Java Version 1.5" section on page 2-2).

Installation Package

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

This section consists of these topics:

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 or 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

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 Version 1.5

To use the Service Configuration API with Java Version 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 Cisco SCE platform to allow correct Service Configuration API functioning.

This section consists of these topics:

Cisco SCA BB Setup

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

Proprietary Remote Procedure Call 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 Guide*.

Before using the Service Configuration API, ensure that:

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

Programming with the SCA BB Service Configuration API

Revised: May 27, 2013, OL-29109-01

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.

This chapter consists of these sections:

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 the Policy and Subscriber classes use.
  - *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, see the Javadoc that is part of the Service Configuration API distribution (see the "Distribution Content" section on 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. This API also provides methods for importing and exporting parts of service configurations 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. See the "Class SCABB Methods" section on page 3-2 for details.

Class SCABB Methods

Class SCABB methods are explained in the following sections:

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 all the API methods that affect the device use.

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

Parameters

- hostName—SCE host address.
- userName—Username.
- password—Password.
- deviceType—Device type. Use the *Connection.SE_DEVICE* device type to connect to an SCE platform.

Return Value

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

Exceptions

This method throws the ConnectionFailedException if the login fails; you can recover 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 all the API methods which affect the device use.

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

Parameters

sessionObject—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—Connection that keeps a handle to the connection to the SCE

addNotificationListener

Syntax

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

Description

Subscribes the specified Cisco SCA BB notification listener to receive notifications about all the operations performed by the Cisco SCA BB API.

The listener must implement the NotificationListener interface.

Parameters

- listener—Notification listener

removeNotificationListener

Syntax

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

Description

Unsubscribes the SCABB listener from receiving notifications about Cisco SCABB API operations.

The removed listener does not receive any more notifications.

Parameters

listener—NotificationListener to be removed

Exceptions

This method throws the IllegalArgumentException if the listener was never subscribed to listen to Cisco 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 all the Service Configuration API and Subscriber API methods use.

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 is imported and exported in XML format.

This section consists of these topics:

Class ImportExportApi Constructor

The Class ImportExportApi constructor is explained in this section:

ImportExportApi

Syntax

```
public ImportExportApi()
```

Description

The ImportExportApi constructor.

Class ImportExportApi Methods

Class ImportExportApi methods are explained in these sections:

exportServiceConfiguration

Syntax

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

Description

Exports a service configuration to a file.

Parameters

- servConf—Service configuration to export
- f—File to which to export the service configuration

Exceptions

This method may throw the following exceptions:

- FileNotFoundException
- ImportExportException—If an error occurs during the export.

importServiceConfiguration

Syntax

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

Description

Imports a service configuration from the specified file.

Parameters

- f—File containing the service configuration to import.

Return Value

The imported service configuration.

Exceptions

This method may throw the following exceptions:

- ImportExportException—If an error occurs during the import.
- IOException

importFlavors

Syntax

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

Description

Imports flavors of a specified type from a specified CSV file.

Parameters

- servConf—Service configuration into which to import the flavors
- flavorType—Type of the imported flavors
- file—CSV file from which to import

Exceptions

This method may throw the following exception:

- ImportExportException—If an error occurs during the import.

importFlavors

Syntax

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

Description

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

Parameters

- servConf—Service configuration into which to import the flavors.
- flavorType—Type of the imported flavors.
- inStream—Input stream from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importFlavorsFromEasyFormat

Syntax

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

Description

Imports flavors of a specified type from a specified CSV file in an Easy Format.

Parameters

- servConf—Service configuration into which to import the flavors.
- file—CSV file from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importHttpRefererFlavorsFromEasyFormat

Syntax

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

Description

Imports flavors of an HTTP Referer from a specified CSV file in Easy format.

Parameters

- servConf—Service configuration into which to import the flavors.

- file—CSV file from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importZones

Syntax

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

Description

Imports zones from a specified CSV file.

Parameters

- servConf—Service configuration into which to import the zones.
- file—CSV file from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importZones

Syntax

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

Description

Imports zones from a given input stream.

Parameters

- servConf—Service configuration into which to import the zones.
- inStream—Input stream from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during import.

importZonesFromEasyFormat

Syntax

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

Description

Imports zones from a specified CSV file in Easy format.

Parameters

- servConf—Service configuration into which to import the zones.
- file—CSV file from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importProtocols

Syntax

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

Description

Imports protocols from a specified CSV file.

Parameters

- servConf—Service configuration into which to import the protocols.
- file—CSV file from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importProtocols

Syntax

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

Description

Imports protocols from a given input stream.

Parameters

- servConf—Service configuration into which to import the protocols.
- inStream—Input stream from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importServices

Syntax

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

Description

Imports services from a specified CSV file.

Parameters

- servConf—Service configuration into which to import the services.
- file—CSV file from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

importServices

Syntax

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

Description

Imports services from a given input stream.

Parameters

- servConf—Service configuration into which to import the services.
- inStream—Input stream from which to import.

Exceptions

This method may throw the ImportExportException if an error occurs during the import.

exportProtocols

Syntax

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

Description

Exports protocols to a specified file in a CSV format.

Parameters

- protocols—List of protocols to export.
- file—File to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportProtocols

Syntax

```
public static void exportProtocols(List protocols,
                                   OutputStream outStream)
throws ImportExportException
```

Description

Exports protocols to a given output stream in a CSV format.

Parameters

- protocols—List of protocols to export.
- outStream—Output stream to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportZones

Syntax

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

Description

Exports zones to a specified file in a CSV format.

Parameters

- zones—List of zones to export.
- file—File to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportZones

Syntax

```
public static void exportZones(List zones,
                               OutputStream outStream)
throws ImportExportException
```

Description

Exports zones to a given output stream in a CSV format.

Parameters

- zones—List of zones to export.
- outStream—Output stream to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportZonesToEasyFormat

Syntax

```
public static void exportZonesToEasyFormat(List zones,
                                File file)
throws ImportExportException
```

Description

Exports zones to a specified CSV file in Easy format.

Parameters

- zones—List of zones to export.
- file—File to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportFlavors

Syntax

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

Description

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

Parameters

- flavors—List of flavors to export.
- flavorType—Type of the exported flavors.
- file—File to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportFlavors

Syntax

```
public static void exportFlavors(List flavors,
                                 FlavorType flavorType,
                                 OutputStream outStream)
throws ImportExportException
```

Description

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

Parameters

- flavors—List of flavors to export.
- flavorType—Type of the exported flavors.
- outStream—Output stream to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportFlavorsToEasyFormat

Syntax

```
public static void exportFlavorsToEasyFormat(List flavors,
                                    File file)
throws ImportExportException
```

Description

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

Parameters

- flavors—List of flavors to export.
- file—File to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportServices

Syntax

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

Description

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

Parameters

- servConf—Service configuration from which to export the services.
- file—File to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

exportServices

Syntax

```
public static void exportServices(ServiceConfig servConf,
                                  OutputStream outStream)
throws ImportExportException
```

Description

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

Parameters

- servConf—Service configuration from which to export the services.
- outStream—Output stream to which to export.

Exceptions

This method may throw the ImportExportException if an error occurs during the export.

loadListArray

Syntax

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

Description

Deprecated—Used only 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—Service configuration into which to import the services.
- inStream—Input stream to import to.

Exceptions

- This method may throw the 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 these sections:

applyServiceConfiguration

Syntax

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

Description

Applies a service configuration to the specified SCE.

Parameters

- connectionApi—Connection API that keeps a handle to the connection to the SCE.
- servConf—Service configuration to apply.

Return Value

The operation time stamp.

Exceptions

This method may throw the following exceptions:

- 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—Connection API that keeps a handle to the connection to the SCE.
- servConf—Service configuration to apply.
- applySettings—Properties.

Return Value

The operation time stamp.

Exceptions

This method may throw the following exceptions:

- 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—Connection API that keeps a handle to the connection to the SCE.
- servConf—Service configuration to apply.
- updateCm—Whether the Collection Manager for the specified SCE will be updated with the specified service configuration values.
- cmIpRemap—Map from the Collection Manager IP address as configured in the SCE, to the actual CM addresses.
- cmUpdateMethod—Method to use to connect to the Collection Manager.
- rpcPort—Port number for the Collection Manager RPC connection
- forceTemplateVirtualLinksInVlMode

Return Value

The operation time stamp.

Exceptions

This method may throw the following exceptions:

- 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—ConnectionApi that keeps a handle to the connection to the SCE.

Return Value

The service configuration in the SCE platform.

Exceptions

This method may throw the following exceptions:

- 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 service configuration.

Parameters

- cmAddress—Address of the CM to update.
- sceAddress—Address of the SCE platform that enforces the given service configuration.
- servConf—Service configuration.

Exceptions

This method may throw the 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 service configuration.

Parameters

- cmAddress—Address of the CM to update.
- sceAddress—Address of the SCE platform that enforces the given service configuration.
- servConf—Service configuration.
- cmUpdateMethod
- rpcPort

Exceptions

This method may throw the 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 about 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—PQB file to load.

Return Value

The resulting service configuration.

Exceptions

This method may throw the following exceptions:

- 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—Service configuration to save.
- pqbfile—PQB file into which to save the service configuration.

Exceptions

This method may throw the following exceptions:

- FileNotFoundException
- ImportExportException

importDefaultServConf

Syntax

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

Description

Loads the default service configuration.

Return Value

The default service configuration

Exceptions

This method may throw the 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 SCE platforms and sets their application parameters to regulate subscriber flows.

Class ServiceConfig Methods

Class ServiceConfig methods are explained in these sections:

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 the service configuration package list.

Return Value

The service configuration package list.

getPolicySettings

Syntax

```
public PolicySettings getPolicySettings()
```

Description

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

Return Value

The service configuration settings for this service.

getProtocolList

Syntax

```
public ProtocolList getProtocolList()
```

Description

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

Return Value

This service configuration protocols list.

getProtocolRedirectIndexNameArray

Syntax

```
public ProtocolRedirectIndexNameArray getProtocolRedirectIndexNameArray()
```

Description

Gets the list of redirect index names of the protocols.

Return Value

The list of redirect index names of the protocols.

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 specification, and exists for only a few predefined protocols.

Parameters

- protocolName—Queried protocol.
- redirectIndex—Index in the redirect String array.

Return Value

The redirect address.

Exceptions

This method may throw the ItemNotFoundException if there is no such predefined protocol in this service configuration 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 alias for the TimeFrame.

Return Value

The TimeFrame API name.

Exceptions

This method may throw the ItemNotFoundException if there is no such alias in this service configuration.

getServiceList

Syntax

```
public ServiceArray getServiceList()
```

Description

Gets this service configuration service list.

Return Value

This service configuration 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 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 specification, and exists for only a few predefined protocols.

Parameters

- protocolName—Queried protocol.

Return Value

The redirect address.

Exceptions

This method may throw the ItemNotFoundException if there is no such predefined protocol in this service configuration ProtocolArray.

getZoneList

Syntax

```
public ZoneList getZoneList()
```

Description

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

Return Value

This service configuration 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 specification, and exists for only a few predefined protocols.

Parameters

- protocolName—Queried protocol name.

Return Value

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

Exceptions

This method may throw the ItemNotFoundException if there is no such predefined protocol in this service configuration 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—Redirects the activity of this protocol.
- redirectIndex—Redirect String index in the protocol redirect String array.
- value—Protocol redirect address.

Exceptions

This method may throw the following exceptions:

- ItemNotFoundException—If there is no such predefined protocol in this service configuration 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—Redirects the activity of this protocol.
- value—Protocol redirect address.

Exceptions

This method may throw the following exceptions:

- ItemNotFoundException—If there is no such predefined protocol in this service configuration 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—Index of the TimeFrame to which the alias is given.
- newName—Alias for the name.

Exceptions

This method may throw the following exceptions:

- 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—TimeFrame to which the alias is given.
- newName—Alias for the name.

Exceptions

This method may throw the following exceptions:

- 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 the connections you create by using any of the login() methods are properly closed by using logout() (see "logout" section on page 3-3).

Service Configuration API Code Examples

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

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.
 * &lt;p&gt;
 * usage: java examples.SimpleApplyPqb &lt;sce-address&gt; &lt;password&gt;
 * &lt;pqb-filename&gt;
 */
public class SimpleApplyPqb {

   public static void main(String[] args) {

      if (args.length != 3) {
         System.err.println("usage: java examples.SimpleApplyPqb "
                        + "&lt;sce-address&gt; &lt;password&gt; &lt;pqb-filename&gt;");
         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 (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.
 * &lt;p&gt;
 * usage: java examples.UpdateZoneFromCsv &lt;sce-address&gt; &lt;password&gt;
 * &lt;zone-csv-file&gt; &lt;zone-name&gt;
 *
 */
public class UpdateZoneFromCsv {

   public static void main(String[] args) {

      if (args.length != 4) {
         System.err.println("usage: java examples.UpdateZoneFromCsv"
                        + " &lt;sce-address&gt; &lt;password&gt;"
                        + " &lt;zone-csv-file&gt; &lt;zone-name&gt;");
         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 &gt; 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());

      }
   }
}
```

Logging and Troubleshooting

Revised: May 27, 2013, OL-29109-01

Introduction

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

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. The log4J enables SCE to log every activated operation in to 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 CLASSPATH. The file is read at startup of the application. If you modify the properties file, you must restart the application.

The following example 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
```