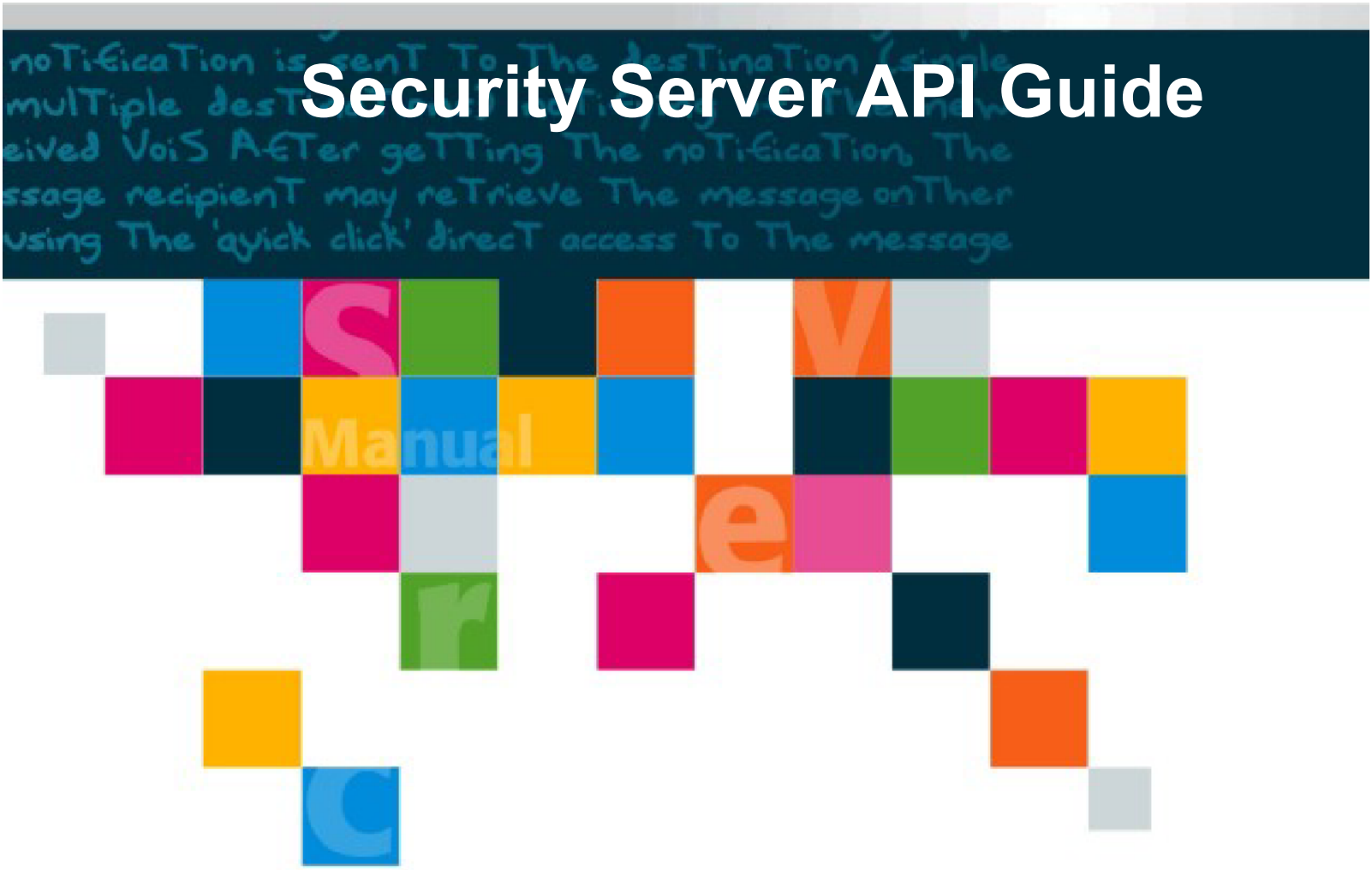
COMVERSE

ComverseONE™

3.5.50

Security Server API Guide



ComONE-3.5.50-SSAG-2010-10-15

Notice

Corporate Headquarters
200 Quannapowitt Parkway
Wakefield, MA 01880 USA
Tel: (781) 246-9000
Fax: (781) 224-8143
www.comverse.com

Revision History

The following table lists the document changes since the initial publication:

Date	Chapter	Description
10/15/2010		Initial publication for the 3.5.50 release.

Contents

Figures

Tables

Notational Conventions

Useful information appears in this format.

Provides direction to important information

Important information appears in this format.

Indicates possible risk of damage to data, software, or hardware.

Indicates serious risk of damage to data, software, or hardware.

Table 1 Notational Conventions

Notation	Explanation of Convention
References to printed documents	*Helvetica italic* **Example:** See *Database Reference Volume 2*.
<KEYS>	UPPERCASE HELVETICA, in angle brackets **Example:** Press <CTRL><Q><SHIFT><P> to create an em dash.
`User-entered text`	**`Courier bold`** **Example:** Enter **`Total Charges`** in the field.
`Placeholders for user-determined text`	*`Courier italic`*, in angle brackets **Example:** Enter your *<password>*.
`Code samples, TABLE_NAMES, field_names, file and directory names, file contents, user names, passwords, UNIX ENVIRONMENT_VARIABLES`	`Courier`
Placeholders for system-generated text	*Helvetica italic* **Example:** Messages appear in this form: *timestamp messageID >> text.*
Buttons, Icon Names, and Menu items	**Helvetica bold** **Example:** Choose **Reports** from the main menu.

Special Markers

The Comverse ONE Billing and Active Customer Management solution has the three derivatives shown in Table 2, "Labels in Markers." For user convenience, any content that is specifically included in a derivative is highlighted with special markers so that it can readily be distinguished.

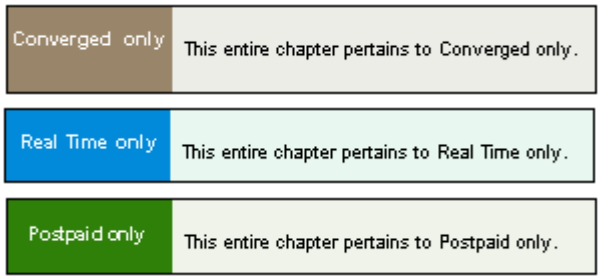
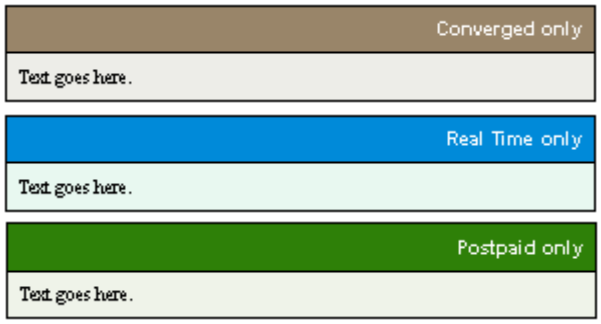
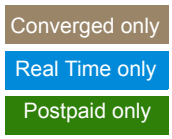
Table 2 Labels in Markers

Derivative	Label Shown in Markers
Comverse ONE Converged Billing derivative	**Converged only**
Comverse ONE Real-Time Charging derivative	**Real Time only**
Comverse ONE Postpaid Billing derivative	**Postpaid only**

Each derivative has a set of three color-coded markers, as shown in Table 3, "Types of Markers." The markers are used individually or in combination to highlight derivative-specific content by:

- Entire chapters
- Selected portions of chapters
- Tables, either entire or partial

Table 3 Types of Markers

Marker	Example	Description
Alert		■ Placed at the beginning of an entire chapter that pertains only to a specific derivative. ■ Placed just before a table that partially or entirely pertains only to a specific derivative.
Block		A shaded box that encloses sections of documentation that pertain only to a specific derivative.
Flag		■ Designates a shaded table row whose contents pertain only to a specific derivative. ■ In a bulleted list, designates an item that pertains only to a specific derivative.

Comverse ONE Documentation List

this is a comprehensive list. As such, it may include documentation for products which you have not licensed.

The documents described below reference the Comverse ONE solution products. All documentation available with the Comverse ONE solution is described in the following pages, organized by the following categories:

- Infrastructure Domain
- Rating, Charging, and Promotions Domain
- Billing and Financials Domain (Converged only)
- Customer and Order Management Domain (Converged only)
  - Customer Relationship Management
    (Sales Force Automation, Case Management, Campaign Management)
- Mediation and Roaming Solutions Domain
- Self-Service Solutions Domain

Read the relevant Solution Description first to get an overview of your Comverse ONE solution. It gives an overview of the functionality in each product domain and also includes cross-references to the user documentation that provides more detailed information about the functionality.
There are two such documents and they are listed under the Infrastructure Domain heading below.

- *Converged Billing & Active Customer Management Solution Description*
- *Real-Time Billing & Active Customer Management Solution Description*

Infrastructure Domain

Download every document in the Infrastructure domain if you purchase the Comverse ONE solution. Documentation for this domain includes the following (in alphabetical order):

- *Alarms Reference*
  Contains tables of alarm IDs, descriptions, likely causes, and recommended resolutions for systems and components.

- *Back Office Administration GUI Guide*
  Provides information about the BackOffice subsystems for Inventory Administration, Address Management and Bulk Operations.
- *Converged Billing & Active Customer Management Solution Description*
  General overview of the Comverse ONE Converged Offer and the functionality available in each domain.
- *Database Reference*
  Describes all database tables and fields in detail.
- *Disaster Recovery Operations Guide* (Optional Module)
  The Disaster Recovery Operations Guide serves as both a technical overview of the optional Disaster Recovery solution and as a guide which details the operational procedures for failover, switchover and switchback provided by the solution.
- *Glossary*
  Provides a list of terms used specifically for the Comverse ONE solution
- *Investigation Units and Financial GUIs Guide*
  Describes the GUI-based tools used for investigating and troubleshooting various financials related processes: payments, bill invoices, refunds, and incomplete data work entries
- *Operation Reference*
  Describes the processes in the Comverse ONE solution.
- *Platform Operations Guide*
  Describes the back-end operations and maintenance functionality of the core Comverse ONE solution components. Includes AIX/HACMP platform and cluster operations, Linux/Veritas platform and cluster operations, backup/recovery, shared storage and fiber switch operations, and tape backup operations.
- *Product Catalog Overview*
  Provides a high-level description of the Comverse ONE solution Product Catalog, which is the primary mechanism for creating, configuring, managing, and propagating Product Catalog versions.
- *Product Catalog User Guide*
  Instructions on using the Product Catalog application to define and manage all aspects of Service provisioning.
- *Real-Time Billing & Active Customer Management Description*
  General overview of the Comverse ONE Real-Time Offer and the functionality available in each domain.
- *Schedulable Entity Reference Manual*
  Documents all the jobs, monitors, and workflows, for each component.
- *Security Platform Operations Guide*
  Technical overview of the security platform and information on how to provision and administer the platform.
- *Security Server API Guide*
  Provides an overview of the interfaces exposed by the Java-based Security SDK API, which client applications can leverage to access various security services, such as authentication, authorization, auditing, key management, and credentials management. Also provides information on the Security Web Services API, which provides interfaces to a subset of Security Server commands (Identity Management commands).
- *Signaling Gateway Unit Guide*
  Describes the hardware, installation, configuration, and maintenance of the Signaling Gateway Unit (SGU) used to connect Comverse real-time systems to the SS7 signaling network using either traditional SS7 protocols or Sigtran (SS7 over IP).
- *System Measurements Guide*
  The Comverse ONE Solution automatically collects statistical data from the Service Logic

Unit (SLU) and the Service Gateway Unit (SGU). This includes service statistics on the SLF layer and platform data on the IPF layer.
This guide describes the format and location of this measurement information and provides a description of the meaning of the data.The measurement data can be used to create reports. It can also be imported into other applications (such as Excel) to be viewed.

- *Unified API Guide*
  General overview of the Unified API, a brief description of its architecture, and information about:
  - Framework classes and the functionality they provide
  - Two standard interfaces provided with the Unified API (client SDK and web services)
  - A subset of Unified API business methods most commonly used
- *Unified Platform Guide*
  Technical overview of the Unified Platform and information on the procedures to manage core systems operations in the Comverse ONE solution.

Rating, Charging, and Promotions Domain

Documentation for this domain includes the following (in alphabetical order):

- *Bulk Provisioning Guide*
  - The *CC Batch* utility enables bulk creation of recharge vouchers and subscribers.
  - The *Bulk Provisioning* Utility enables bulk creation of anonymous accounts to support the pre-activation of pre-paid SIM cards.
- *Call Flows Reference*
  Call flows detail the logic flow of specific scenarios. Multiple access numbers can map to the same call flow. Different resellers have the option to publish different numbers but share the same logic.
- *Charging Interfaces Guide*
  Describes the four interfaces that enable external services to support real-time authorization, rating, and charging for transactional usage: (1) the Event Charging Interface, a simple TCP/IP-based interface, (2) Open Services Access (OSA), (3) a Diameter-based interface version enhanced to take advantage of features of the Comverse ONE solution, and (4) a Diameter-based interface packet-switched version.
- *Customer Care Client Provisioning Guide* — Real-Time
  Detailed task-oriented instructions for using Customer Care Client.
- *Diameter Gateway Unit Guide*
  Describes the hardware, installation, configuration and maintenance of the Diameter Gateway Unit (DGU) used to connect Comverse real-time systems to external services, using the diameter protocol over IP.
- *Network Interfaces and Notifications Guide*
  Describes the operation, features, and provisioning of notifications, CAMEL-enabled services, and USSD-enabled services.
- *Network Self-Care Guide*
  Describes the configuration, structure, and features.
- *Rating Technical Reference*
  Describes the Unified Rating Engine, which is the subsystem responsible for gathering incoming CDRs and processing them for billing.
- *Reports and Data Extracts Guide* — Real-Time
  Describes the real-time Operational Reports Interface (ORI) and the Data Warehouse Extract Utility.

- *Recurring–Non-Recurring Charges Server Guide*
  Describes all processes commonly available through the Recurring —Non-Recurring Charges Server.
- *Voucher and Recharge Guide*
  Describes the process by which subscribers add funds to accounts using recharge vouchers through IVR, interaction with Customer Service, and other methods. Provides details of the Recharge Control Table, which allows resellers to provision the effects of recharges so that bonuses, discounts, and other changes to offers can result from a successful recharge. Also describes the Card Generator software used to create batches of recharge vouchers.

Billing and Financials Domain (*Converged* only)

Documentation for this domain includes the following (in alphabetical order):

- *Advanced Statement Numbering Guide*
  Describes how to configure and use Advanced Statement Numbering.
- *Billing Reports and File Layouts User Guide*
  Describes control reports and other file formats.
- *Billing Technical Reference*
  High-level descriptions of billing architecture, administration, bill generation and formatting, and system parameters
- *Collections Guide*
  Contains information on configuring Collections database tables, running the Collections module, and using the Collections interface.
- *Invoice Designer Strings and Filters Reference*
  Describes the static strings, dynamic strings, and filters in the Invoice Designer.
- *Invoice Designer Technical Reference*
  Describes how to configure and run Invoice Designer.
- *Invoice Designer User Guide*
  Describes the Invoice Designer and how to perform the tasks needed to create an invoice template.
- *Journals Guide*
  Describes the theory, configuration, and running of Journals processes.
- *Miscellaneous Configurable Entities*
  Instructions for configuring late fees, adjustments, and several other database entities used in postpaid and converged billing.
- *Process Workflow Orchestration Guide*
  Describes the command-line entries and the default queries for running billing-related processes via the Unified Platform.
- *Taxation Guide*
  Describes the configuration, operation, structure, and features of Taxation.

Customer and Order Management Domain (*Converged* only)

Documentation for this domain includes the following (in alphabetical order):

- *Application Integrator Adapter Developer Kit User Guide*
  Provides information necessary for the development of custom Application Integrator adapters.
- *Application Integrator Add/Copy Header User Guide*
  Describes the adapter that adds or copies header information in messages.
- *Application Integrator Aggregator Adapter User Guide*
  Describes the adapter that aggregates multiple input messages as a single composite output message.
- *Application Integrator File Adapter User Guide*
  Describes the configuration process and rules for the file adapter.
- *Application Integrator CORBA Adapter (JacORB) User Guide*
  Describes the elements and uses of the Application Integrator client and server Common Object Request Broker Architecture (CORBA) adapters for JacORB.
- *Application Integrator Filemover Adapter User Guide*
  Describes the use and configuration of the adapter, which is used to copy or move files from one machine to another.
- *Application Integrator Generic Services User Guide*
  Describes the Null adapter, Trash adapter, and Initiator adapter generic services.
- *Application Integrator HTTP Adapter User Guide*
  Describes the use and configuration of the adapter which provides an interface between HTTP clients and the ApplicationIntegrator.
- *Application Integrator IPDR Adapter User Guide*
  Describes use and configuration of the I adapter which converts the “compact encoding” form of IPDR billing record documents into a form easily parsed by the ApplicationIntegrator message broker.
- *Application Integrator JMS Adapter User Guide*
  Describes the use and configuration of the adapter, which is used with edge systems that transmit or receive JMS messages.
- *Application Integrator KSI Adapter User Guide*
  Describes the use and configuration of the adapter, which is used with edge systems that transmit or receive data formatted according to the Kenan Standard Interface (KSI) protocol.
- *Application Integrator Operator Guide*
  Describes the commands that operate the Application Integrator at creation and runtime.
- *Application Integrator Python Adapter User Guide*
  Describes the use and configuration of the adapter, which enables a user to run a *Python* script from within an integration.
- *Application Integrator Retry Adapter User Guide*
  Describes the use of the a dapter to resend messages in case of failed transmissions.
- *Application Integrator SAS Adapter User Guide*
  Describes the use and configuration of the adapter, which is used with edge systems that transmit or receive data formatted according to the *Comptel* Mediation Device Solutions/Subscriber Administration System (MDS/SAS) protocol.
- *Application Integrator Sequence Adapter User Guide*
  Describes the use of the adapter to generate unique sequence numbers for messages.
- *Application Integrator System Administrator Guide*
  Outlines installation, sizing, operation, and administration of the Application Integrator

and logging. Describes configuration of the user environment and commands for creation and operation of the Application Integrator.

- *Application Integrator Unified API Client Adapter User Guide*
  Describes the adapter which is used for interfaces based on the Unified API Client.
- *Application Integrator Unified API Server Adapter User Guide*
  Describes the adapter ehich is used for interfaces based on the Unified API Server.
- *Application Integrator URL Client Adapter User Guide*
  Describes the use and confoguration of the adapter which makes it possible for a client to gain access to many kinds of network-accessible resources that are identified by a URL.
- *Application Integrator User Guide*
  Describes creating integration specifications, creating instances of the Application Integrator, and commands for operation of the Application Integrator. Provides a complete user guide for the iMaker compiler.
- *Application Integrator XSLT User Guide*
  Describes the use and configuration of the adapter which is usedwith applications (sometimes called edge systems) that transmit or receive XML- formatted data.
- *Customer Center User Guide*
  Detailed task-oriented instructions for using Customer Center.
- *Inventory Guide*
  Describes the configuration, operation, structure, and features of Inventory.
- *Inventory Replenishment Guide*
  Describes the operation, structure, and features of Inventory Replenishment.
- *Orders Services Guide*
  Describes the structure and features of Orders Services.
- *Request Handling and Tracking and Service Fulfillment User Guide*
  Describes the configuration, operation, structure and features of Request Handling and Tracking and Service Fulfillment.
- *Workflow Developers Guide*
  Helps new users understand the rules-based business process management system so users can create solutions and integrate Workpoint within those solutions.
- *Workflow User Guide*
  Describes the configuration, operation, structure, and features of Workpoint.

Customer Relationship Management

- *Billing Reports and File Layouts User Guide*
  Describes control reports and other file formats.
- *Campaign Management Data Mapping Reference*
  Describes how the data in DataMart is mapped to information in the Comverse ONE Customer database, the Comverse ONE ODS, and the Comverse ONE Sales and Service database.
- *Campaign Management DataMart Reference*
  Contains in-depth technical information on how to configure and populate the data mart used by all Campaign Management applications.
- *Campaign Management Outbound Marketing Manager Reference*
  Describes how to use the Campaign Management Outbound Marketing Manager features and guides you through the program's basic functionality.
- *Campaign Management Quick Implementation Guide*
  Helps novice users get started with implementing Campaign Management. It contains an overview of the product architecture, information on data mart design and creation, an explanation of how extraction works, and procedures for creating web pages, reports, lists, and campaigns.
- *Campaign Management Topic Implementation Guide*
  Provides information for implementers and professional services personnel who are creating applications that will run on an Campaign Management EpiCenter. Summarizes the Campaign Management functionality, architecture, and administration and contains in-depth technical information for configuring the Campaign Management topics required for Campaign Management and analysis.
- *Campaign Management User Guide*
  Provides you with basic information about the Campaign Management applications.
- *Case Management User and Administration Guide*
  Contains detailed information about GUI screens and form fields that appear in the Case Management application. Also provides information on performing general procedures in the GUI and administrative tasks.
- *Customer Center User Guide*
  Detailed task-oriented instructions for using Customer Center.
- *Sales and Service Admin Console User Guide*
  Provides supervisors, managers, and executives with the information to use the Case Management and Sales Force Automation Admin Console application.
- *Sales and Service Application Reference* Contains technical reference information relevant to implementers involved in implementing and customizing CRM applications at customer sites. This book provides the reference context for the procedural information available in the Implementation Guide.
- *Sales and Service Architecture Reference*
  Provides technical information relevant to individuals involved in implementing the Open Architecture and the applications built on the architecture
- *Sales and Service Data Dictionary Reference*
  Includes a listing and description of the tables and columns used to store CRM operational business data. It also includes a description of the naming conventions for the tables. The target audience includes database administrators, application developers, and implementers.
- *Sales and Service IBR Designer User Guide*
  Describes how to use the IBR Designer to create Intelligent Business Rules, which can be used to implement rule-based behavior within your CRM applications.

- *Sales and Service Implementation Guide*
  Provides procedural information relevant to individuals involved in implementing and customizing the core and the Sales and Service applications built on the core.
- *Sales and Service Integration Guide*
  Provides overview and configuration information for the set of tools used to exchange data with a variety of back-end data sources, including generic SQL sources, Java and EJB-based sources, Web services, and other database types.
- *Sales and Service Workflow Designer*
  Explains how to use Workflow Designer, a web-based graphical tool for defining and editing workflows
- *Sales Force Automation User and Administration Guide*
  Contains detailed information about GUI screens and form fields that appear in the Sales Force Automation application. Also provides information on performing general procedures in the GUI and administrative tasks.

Mediation and Roaming Solutions Domain

Documentation for this domain is subdivided into Mediation/Roaming and Revenue Settlements.

Mediation and Roaming

Mediation and Roaming documentation includes the following (in alphabetical order):

- *Collection API Guide*
  Provides the concepts and functions for the Collection Application Programming Interface (CAPI).
- *Data Manager GUI Reference*
  Contains detailed information about GUI screens and form fields that appear in the Data Manager interface
- *GRID Mapping Language Developer Guide*
  Describes the mediation feature components, semantics, and general syntax of the GRID Mapping Language (GML).
- *Installation Guide for HP*
  Describes how to install and configure the application, components, and some third-party applications associated with the HP platform.
- *Installation Guide for HP Itanium*
  Describes how to install and configure the application, components, and some third-party applications associated with the HP Itanium platform.
- *Installation Guide for HP PA-RISC*
  Describes how to install and configure the application, components, and some third-party applications associated with the HP PA-RISC platform.
- *Installation Guide for IBM*
  Describes how to install and configure the application, components, and some third-party applications associated with the IBM platform.
- *Installation Guide for SUN*
  Describes how to install and configure the application, components, and some third-party applications associated with the SUN platform.
- *Mediation and Roaming User Guide*
  Provides information on how to use the GUI interface, including information on using the Data System Manager application pages.
- *Mediation API Guide*
  Contains reference information on using the Mediation API.

- *Roaming Database Reference*
  Provides reference information on the Roaming database.
- *Roaming Setup Guide*
  Describes how to configure the Roaming Setup application pages. It also provides information on working with TAP, RAP, and CIBER statistics.
- *Scripts Guide*
  Provides information on script files, which contain additional instructions on functions for data collection and transmission.
- *Socket-Based API Guide*
  Explains the building applications using the Socket-Based Record Transmission API. Programmers can use the guide to use the records received from the Data system for their own customized downstream application solutions.
- *System Manager GUI Reference*
  Contains detailed information about GUI screens and form fields that appear in the System Manager interface
- *Variable-Length GRID Guide*
  Provides information on how to configure the control files for variable-length GRID.

Revenue Settlements

Revenue Settlements documentation includes the following (in alphabetical order):

- *Comverse Revenue Settlements Billing System Adapter Guide*
  Describes the configuration, operation, and installation for the Billing System adapter.
- *Comverse Revenue Settlements Data Model Guide*
  Overview of data model entities (such as partners, accounts, revenue sharing, and rate schedules) and how to configure them in the database.
- *Comverse Revenue Settlements Database Reference*
  Detailed descriptions of fields and tables in the database.
- *Comverse Revenue Settlements Technical Reference*
  Instructions for installing and operating Revenue Settlements. Also contains processing descriptions.
- *Comverse Revenue Settlements User Guide*
  Instructions for using the Revenue Settlements GUI.

Self-Service Solutions Domain

The Comverse ONE Self-Service Solutions domain consists of the core products plus the optional separately licensed premium products. The core products consist of the following:

- Self-Service Solutions Platform
- Self-Service Solutions Applications

Self-Service Solutions Platform Documentation

The Self-Service Solutions Platform has a comprehensive set of documentation covering the installation, configuration, and use of our products. The documentation set is divided into the following categories:

- **Manuals**: These manuals cover installing and using the platform.
- **Reference**: These reference documents contain information about APIs, databases, configuration files, and so on. These documents are delivered in HTML.

Self-Service Solutions Platform Manuals

Self-Service Solutions Platform manuals include the following (in alphabetical order):

- *Administration Guide*
  Provides operations and maintenance instructions for Web applications using the Self-Service Solutions Platform.
- *Communications Billing and Usage Reference*
  Provides detailed descriptions of the data models and structure of the Self-Service Solutions Platform Communications Billing and Usage (CBU) database.
- *Connectors Development Guide*
  Provides instructions for developing and customizing Connectors of the Self-Service Solutions Platform.
- *Core Module Development Guide*
  Provides instructions for configuring and developing features of the core module of the Self-Service Solutions Platform.
- *Customer Interaction Datastore Reference*
  Provides detailed descriptions of the data models and the structure of the Self-Service Solutions Platform Customer Interaction Datastore (CID).
- *Database Modules Development Guide*
  Provides instructions for configuring, customizing, and developing features of the database module of the Self-Service Solutions Platform.
- *Platform Installation Guide*
  Provides installation and configuration instructions for the Self-Service Solutions Platform.
- *Platform Services Guide*
  Provides instructions for configuring, customizing, and developing features that use the services provided by the Self-Service Solutions Platform.
- *Processors Development Guide*
  Provides instructions for developing and customizing Processors of the Self-Service Solutions Platform.
- *Reports Development Guide*
  Provides instructions for developing and customizing Reports of the Self-Service Solutions Platform.
- *Self-Service Solutions Overview Guide*
  Provides a high-level architectural and functional description of the Comverse ONE Self-Service Solutions. It also includes a detailed description of the concepts and development process to create and deploy Self-Service Solutions.
- *Web Applications Development Guide*
  Provides instructions for configuring, developing, and deploying Web applications that use the Self-Service Solutions Platform.

Self-Service Solutions Platform Reference

Self-Service Solutions Platform reference documentation includes the following (in alphabetical order):

- *Base Logic Manager Reference*
  Describes usage syntax and configuration files for the Base Logic Manager (BLM) APIs. These APIs are the core services of the Self-Service Solutions Platform.
- *CID2CBU Object Mapping Reference*
  Describes the default mapping of Customer Interaction Datastore (CID) and Communications Billing and Usage (CBU) objects.

- *Communications Billing and Usage Reference*
  Provides detailed descriptions of fields and tables in the Communications Billing and Usage (CBU) database.
- *Customer Interaction Datastore Reference*
  Provides detailed descriptions of fields and tables in the Customer Interaction Datastore (CID).
- *Integration Services Framework API Reference*
  Describes usage syntax of the set of APIs to program connectors and other components of the Intelligent Synchronization Framework (ISF).
- *Integration Services Framework Message Cache Reference*
  Provides detailed descriptions of fields and tables in the Intelligent Synchronization Framework (ISF) Message Cache.
- *Integration Services Framework Script API Reference*
  Describes usage syntax of the Intelligent Synchronization Framework (ISF) script APIs to program the ISF connectors.
- *JavaServer Page Framework for Internet Application API Reference*
  Describes usage syntax for the JavaServer Page Framework for Internet Application (JFN) APIs. These APIs are used to build JSPs using the JFN. This framework provides basic application functions and services as the foundation of user interfaces.
- *Logger Message Reference*
  Provides detailed descriptions of the Self-Service Solutions Platform log messages.
- *QRA API Reference*
  Describes usage syntax for the Query, Reporting, and Analysis (QRA) Engine APIs. These APIs are used to build reports.
- *UTIL API Reference*
  Describes usage syntax for the UTIL package used by different components of the Self-Service Solutions Platform. This package contains a set of utilities including the logger.Self-Service Solutions Applications Documentation

Each Self-Service Solutions Application comes with a comprehensive set of documentation covering the installation, configuration, and use of the product. The application documentation expands and complements the Self-Service Solutions Platform documentation.

The documentation set is divided into the following categories:

- **Manuals**: These manuals cover installing and using the application.
- **Reference**: These reference documents contain information about APIs, databases, configuration files, and so on. These documents are delivered in HTML.

Self-Service Solutions Application Manuals

A full set of these manuals is available for each Self-Service Solutions Application. The documentation set includes the following (in alphabetical order):

- *Business Objects Model Reference*
  Provides a detailed description of the models and entities that make up the Self-Service Solutions Application.
- *Catalog Loader Reference*
  Provides information about the Catalog Loader, including a functional description as well as installation, configuration, and use instructions.
- *Configuration and Development Guide*
  Provides instructions for configuring and developing Self-Service Solutions Application features.
- *Feature Reference*
  Describes the logic and provides use cases for the functional domains of the application.

- *Out-of-the-Box Reference Guide*
  Describes the Self-Service Solutions Application Out-of-the-Box release.
- *Self-Service Installation Guide for Comverse ONE*
  Provides detailed installation, configuration, and deployment instructions for the Self-Service Solutions Application alongside other elements of the Comverse ONE solution.
- *Self-Service Installation and Deployment Guide*
  Provides detailed installation, configuration, and deployment instructions for the Self-Service Solutions Application.
- *Introduction*
  Provides a high-level architectural and functional description of the Self-Service Solutions Application. It covers common features, order management, account management, and bill presentment.

Self-Service Solutions Application References

A full set of these references is available for each Self-Service Solutions Application.The reference documentation set includes the following (in alphabetical order):

- *API Reference*
  Describes usage syntax for the Self-Service Solutions Application APIs. These APIs are used to program the user interface and manage data.
- *Invoice Schema Reference*
  Describes the invoice schema reference of the Self-Service Solutions Application.
- *Presentation Layer Page Flow Reference*
  Describes the page flows of the Self-Service Solutions Application.
- *Specification Entity Relationship Diagrams*
  Provides diagrams describing the actors, use cases, user activity, and storyboard in IBM Rational Rose format.

Self-Service Solutions - Separately Licensed Products

Documentation available with optional, separately-licensed premium products in the Comverse Self-Service Solutions is listed below.

Online Catalog Manager

Online Catalog Manager (OCM) documentation includes the following (in alphabetical order):

- *Introduction to the Online Catalog Manager*
  Provides a high-level architectural and functional description of the Online Catalog Manager.
- *Online Catalog Manager Getting Started Guide*
  Describes the best way to build product catalogs in the Online Catalog Manager. This manual is a template for creating end-user documentation.
- *Online Catalog Manager Installation and Configuration Guide*
  Provides installation and configuration instructions for the Online Catalog Manager.
- *Online Catalog Manager User Documentation Template*
  Describes the use of the Online Catalog Manager. This manual is a template for creating end-user documentation. This manual covers many common concepts and procedures of the OCM.
- *Online Catalog Manager User Guide*
  Provides a detailed description of the concepts and use of the Online Catalog Manager. The topics include:
  - Managing Media Files

- Managing Offers
- Managing Prices
- Managing Products
- Managing Properties
- Managing Reference Data
- Publishing

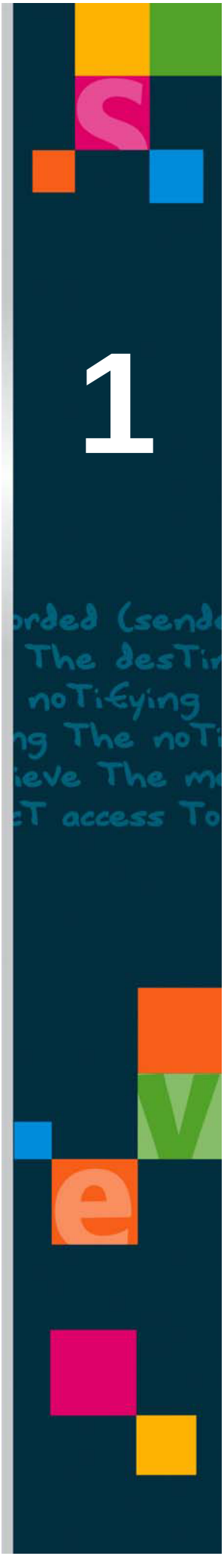
CSR Portal

The CSR Portal product includes the standard Application documentation, plus the following manual:

- *CSR Portal User Guide*
  A guide to using the CSR Portal UI.

Chapter 1

Introduction



Welcome

Welcome to the *Security Server API Guide* for the Comverse ONE Billing and Active Customer Management solution. This document discusses the Application Programming Interfaces (APIs) used by client applications to access security services hosted on the Security Server or to invoke Security Server operations.

New Features for This Release

The following new features in the Comverse ONE 3.5.50 release impact this document:

- There are no new features in the 3.5.50 release that impact this document.

Two Security APIs

The following two Java-based APIs are available and discussed in this document:

- Security SDK API

  Information in this document is intended to convey how applications that are part of the Comverse ONE solution can use the Security SDK API to access security services hosted on the Security Server. Thick clients in the Comverse ONE solution, such as the Product Catalog GUI, use the Security SDK API.

  The Security SDK API abstracts all complex processing logic from client applications. Application developers need to implement only business logic, and the API handles all application-level and transport-level protocols required for communicating with the security services.

  This document provides an overview of all interfaces exposed by the Security SDK API and explains how client applications can leverage these interfaces to access various security services, such as authentication, authorization, auditing, key management, and credentials management.

- Security Server Web Services API

  The Security Server Web Services API provides access to operations for the Security Server. Instead of executing these operations from the Security Server command line interface (CLI) or from the graphical user interface (GUI), the operations can be invoked using the Security Server Web Services API. A subset of operations (Identity Management operations) is exposed through the API. Typically, third-party applications (such as Trivnet), Comverse Self-Service, and customer applications use the Web Services API.

The Security SDK API is primarily for client-based operations, such as login, audit record generation and handling, and so on. With the SDK API, some tasks, such as managing the user session and checking for the security token validity, can be done on the client side. The SDK API also provides the ability to create an embedded Policy Decision Point (PDP), load security authorization policies, and perform related policy functions.

In contrast to the Security SDK API, the Security Server Web Services API is for administrative operations, such as adding users, adding security roles, and similar administrative tasks. The Web Services API supports client-based operations only for identity management. The Web Services API currently provides no client-based access to operations for authorization policy management, credentials management (involving database passwords and SNMP community strings for network devices), or other operations unrelated to identity management.

Who Should Use This Document

This document is intended for use as a reference by developers and system integrators who need to use the security features of the Comverse ONE solution. It is assumed that you are thoroughly familiar with the concepts discussed in the *Security Platform Operations Guide*. In addition, to work with the APIs, you need a solid background in the following:

- Java programming language (Java 1.5 or higher)
- Security Assertion Markup Language (SAML)
- Extensible Access Control Markup Language (XACML)
- Distributed Audit Service (XDAS)
- Web service calls (if using the Security Server Web Services API)

Organization of This Document

The remainder of this document is organized as follows:

Chapter 2, "Security SDK API Overview," provides a general overview of the functional areas covered by the Security SDK API, libraries required by the API, and system properties that govern the behavior of the API and control how it communicates with the Security Server.

Chapter 3, "Security SDK Client API Reference," provides information on the client application interfaces exposed by the Security SDK API.

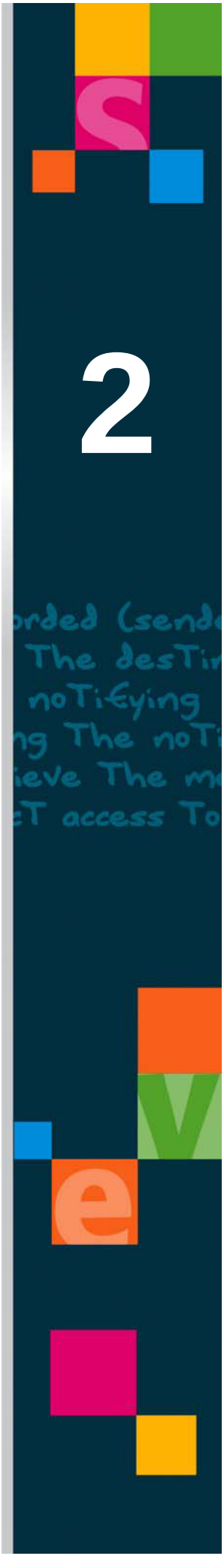
Chapter 4, "Security Server Web Services API," provides information on the Identity Management operations available with the Security Server Web Services API.

Chapter 5, "Security SDK API Code Examples," provides various code examples demonstrating how client applications can use the Security SDK API.

Appendix A, "Attribute Conflict-Resolution Rules," provides an explanation of the rules used to resolve conflicts among custom attributes defined at the security realm, group, and user levels.

Chapter 2

Security SDK API Overview



Functional Areas

The Security SDK API can be categorized into the following five broad functional areas:

- Identity Management and Authentication
- Policy Management and Authorization
- Audit Management
- Credentials Management
- Cryptography and Key Management

The Identity Management and Authentication API provides interfaces for performing all identity management functions for a user such as login, logout, user password changes, attribute management, obtaining the single sign-on (SSO) token, and so on. This API is responsible for invoking web services hosted on the Security Server for authentication purposes and for generating and parsing Security Assertion Markup Language (SAML) request and response messages.

The Policy Management and Authorization API provides interfaces for performing client-side access control functions. This API is responsible for providing client applications an ability to retrieve access control policies (also known as authorization policies) from the Security Server and also for determining if access to a given resource is allowed. Policies are generated and published using the Extensible Access Control Markup Language (XACML).

The Audit Management API provides interfaces for creating, modifying, and publishing audit records to the Unified Platform Agent (UPA). The Audit Management API generates audit records that adhere to the Distributed Audit Service (XDAS) specification and securely transports these records to a local or remote UPA and eventually to the Security Server.

The Credentials Management API provides interfaces for accessing credentials, such as database passwords and SNMP community strings for network devices, from the Security Server.

The Cryptography API provides interfaces that client applications can use to secure confidential data for transport over an unsecured network or to securely store data locally. The Cryptography API facilitates encryption of data using both symmetric and asymmetric algorithms, and it provides mechanisms for digitally signing data, obtaining message digests, and so on. The Key Management API facilitates creation, disabling, and fetching of symmetric keys to be used for encryption from the Security Server. The Key Management API is responsible for secure transmission of symmetric keys between client applications and the Key Management web service hosted on the Security Server. The Key Management API uses a proprietary XML-based protocol to communicate with the corresponding service on the Security Server.

Getting Started

The following sections provide information needed to get started when using the Security SDK API.

JAR Files Required for the Security SDK API

Table 4 on page 8 describes the JAR files required for the Security SDK API. All the JARs must be included in the client application's CLASSPATH. Also see "JAR Files Required for Web Services Stacks" on page 8 for additional information.

Table 4 JAR Files Required

JAR File	Path to Obtain File (Internal Comverse Server Location)
Security SDK Framework JAR	
securityframework-1.0.27.jar	/usr1/arbor/security_repo/com/comverse/security /securityframework/1.0.26/securityframework-1.0.27.jar
Third-Party JARs Required by the Security SDK	
activation-1.1.jar	/usr1/arbor/3p_maven_repo/javax/activation/activation/1.1 /activation-1.1.jar
bcprov-jdk15-135.jar	/usr1/arbor/3p_maven_repo/bouncycastle/bcprov-jdk15/135 /bcprov-jdk15-135.jar
commons-codec-1.3.jar	/usr1/arbor/3p_maven_repo/commons-codec/commons-codec/1.3 /commons-codec-1.3.jar
commons-io-1.3.1.jar	/usr1/arbor/3p_maven_repo/commons-io/commons-io/1.3.1 /commons-io-1.3.1.jar
commons-lang-2.3.jar	/usr1/arbor/3p_maven_repo/commons-lang/commons-lang/2.3 /commons-lang-2.3.jar
commons-logging-1.1.jar	/usr1/arbor/3p_maven_repo/commons-logging/commons-logging/1.1 /commons-logging-1.1.jar
jdom-1.0.jar	/usr1/arbor/3p_maven_repo/jdom/jdom/1.0/jdom-1.0.jar
log4j-1.2.14.jar	/usr1/arbor/3p_maven_repo/log4j/log4j/1.2.14/log4j-1.2.14.jar
mail-1.4.jar	/usr1/arbor/3p_maven_repo/javax/mail/mail/1.4/mail-1.4.jar
ojdbc14-10.2.0.2.jar	/usr1/arbor/3p_maven_repo/ojdbc14/ojdbc14/10.2.0.2 /ojdbc14-10.2.0.2.jar
opensaml-1.1.jar	/usr1/arbor/3p_maven_repo/opensaml/opensaml/1.1 /opensaml-1.1.jar
serializer-1.0.jar	/usr1/arbor/3p_maven_repo/serializer/1.0/serializer-1.0.jar
servlet-api-2.4.jar	/usr1/arbor/3p_maven_repo/javax/servlet/servlet-api/2.4 /servlet-api-2.4.jar
sun-xacml-2.0.jar	/usr1/arbor/3p_maven_repo/sun-xacml/sun-xacml/2.0 /sun-xacml-2.0.jar
xalan-2.7.0.jar	/usr1/arbor/3p_maven_repo/xalan/xalan/2.7.0/xalan-2.7.0.jar
xercesImpl-2.8.0.jar	/usr1/arbor/3p_maven_repo/xerces/xercesImpl/2.8.0 /xercesImpl-2.8.0.jar
xml-apis-1.0.b2.jar	/usr1/arbor/3p_maven_repo/xml-apis/xml-apis/1.0.b2 /xml-apis-1.0.b2.jar
xmlsec-1.3.0.jar	/usr1/arbor/3p_maven_repo/xml-security/xmlsec/1.3.0 /xmlsec-1.3.0.jar
xpp3-1.1.4c.jar	/usr1/arbor/3p_maven_repo/xpp3/xpp3/1.1.4c/xpp3-1.1.4c.jar

JAR Files Required for Web Services Stacks

The following web services stacks (libraries) are supported for client applications that use the Security SDK API:

- JBoss stack
- WebLogic stack
- JWSDP stack

The sections below specify the libraries (JAR files) required for client applications, based on which web services stack the client application uses. All JARs must be included in the client application's CLASSPATH.

JBoss Stack

Client applications using the JBoss stack need the JBoss libraries shown in Table 5, along with the JARs shown in Table 4 on page 8.

Table 5 Additional JAR Files Required for JBoss

JAR File	Path to Obtain File (Internal Comverse Server Location)
jbossall-client-cmvt-4.0.4.jar	/usr1/arbor/3p_maven_repo/jboss/jbossall-client-cmvt/4.0.4 /jbossall-client-cmvt-4.0.4.jar
jbossretro-rt-4.0.4.jar	/usr1/arbor/3p_maven_repo/jboss/retro/4.0.4/jbossretro-rt-4.0.4.jar
jbossws14-client-4.0.4.jar	/usr1/arbor/3p_maven_repo/jboss/webservice/4.0.4 /jbossws14-client-4.0.4.jar
jbossall-client-4.0.4.jar	/usr1/arbor/3p_maven_repo/jboss/jbossall-client/4.0.4 /jbossall-client-4.0.4.jar

WebLogic Stack

Client applications using the WebLogic10 stack need the WebLogic web services client stack libraries shown in Table 6, along with the JARs shown in Table 4 on page 8.

Table 6 Additional JAR Files Required for WebLogic

JAR File	Path to Obtain File (Internal Comverse Server Location)
webserviceclient_ssl-10.0.jar	/usr1/arbor/3p_maven_repo/bea/webserviceclient_ssl/10.0 /webserviceclient_ssl-10.0.jar
wlfullclient-cmvt-10.0.jar	/usr1/arbor/3p_maven_repo/bea/wlfullclient-cmvt/10.0 /wlfullclient-cmvt-10.0.jar

For WebLogic, also set the following system properties:

- Dbea.home=<*location for WebLogic license*>
- Dweblogic.webservice.client.ssl.adapterclass=weblogic.webservice.client.JSSEAdapter
- Dweblogic.webservice.client.ssl.strictcertchecking=false

JWSDP Stack

Client applications using the JWSDP stack need the JWSDP libraries shown in Table 7 on page 10, along with the JARs shown in Table 4 on page 8.

Use the Security SDK version 1.0.7 or higher.

Table 7 Additional JAR Files Required for JWSDP

JAR File	Path to Obtain File (Internal Comverse Server Location)
webservices-tools.jar	/usr1/arbor/3p_maven_repo/javax/jws/webservices-tools/1.1 /webservices-tools-1.1.jar)
webservices-api.jar	/usr1/arbor/3p_maven_repo/javax/jws/webservices-api/1.1 /webservices-api-1.1.jar)
webservices-extra.jar	/usr1/arbor/3p_maven_repo/javax/jws/webservices-extra/1.1 /webservices-extra-1.1.jar)
webservices-extra-api.jar	/usr1/arbor/3p_maven_repo/javax/jws/webservices-extra-api/1.1 /webservices-extra-api-1.1.jar)
webservices-rt.jar	/usr1/arbor/3p_maven_repo/javax/jws/webservices-rt/1.1 /webservices-rt-1.1.jar)

For JWSDP, set the following additional property in your code or as a JVM argument:

- Dsecurity.webservice.stack = jwsdp

Functional Areas for Third-Party Dependent JARs

Table 8 describes the functional areas for the third-party dependent JAR files.

Table 8 Third-Party JAR Files and Functional Areas

JAR Files	Functional Area
sun-xacml-2.0.jar	Authorization
opensaml-1.1.jar	Authentication
bcprov-jdk14-135.jar	Cryptography
jdom-1.0.jar xpp3-1.1.4c.jar	XML parsing
ojdbc14-10.2.0.2.jar	Database connection
commons-codec-1.3.jar commons-lang-2.3.jar commons-io-1.3.1.jar commons-logging-1.1.jar log4j-1.2.14.jar serializer-1.0.jar xalan-2.7.0.jar xercesImpl-2.8.0.jar xml-apis-1.0.b2.jar xmlsec-1.3.0.jar servlet-api-2.4.jar activation-1.1.jar jbossall-client-4.0.4.jar backport-util-concurrent-3.1.jar jbossretro-rt-4.0.4.jar jbossws14-client-4.0.4.jar jbossall-client-cmvt-4.0.4.jar mail-1.4.jar	Dynamic invocation of web services (for the JBoss web services stack)

The `ojdbc14-10.2.0.2.jar` is required only when the Security SDK API is used as a part of a Java stored procedure in the Oracle database. Generally, client applications need not include this file.

By default, the Security SDK API uses the JBoss web services stack for web services communication. If the client application overrides the default property to use other web services libraries (such as the WebLogic stack), then the set of libraries shown in Table 8 can be replaced with the corresponding web services libraries.

System Properties

The Security SDK API refers to a set of system properties that governs the behavior of the API and controls how the API communicates with the Security Server. Table 9 summarizes all the system properties that the Security SDK API refers to. The Category column designates which of the five broad functional areas the property supports. If a property is more global in nature, that is noted in the Category column instead of a specific category. These properties are defined in a properties file, which is unique to each client application. When the client application starts, this properties file is read in and the properties are accessible via Java's system property API.

The Security SDK API assumes default values for all the system properties listed in Table 9. Client applications can override these system properties to suit their needs.

Table 9 Security-Related System Properties

Category	Property	Default	Description
All categories	security.server.ip	secserv	IP address or hostname of the Security Server.
All categories	security.server.peer.ip	peer-secserv	IP address or hostname of the peer Security Server. Used in Disaster Recovery sites to redirect requests to the standby (peer) Security Server in case the production Security Server is down.
All categories	security.sdk.mode	false	If set to `true`, the API returns dummy values. This property can be used to test API interfaces without connecting to the Security Server.

Table 9 Security-Related System Properties (Continued)

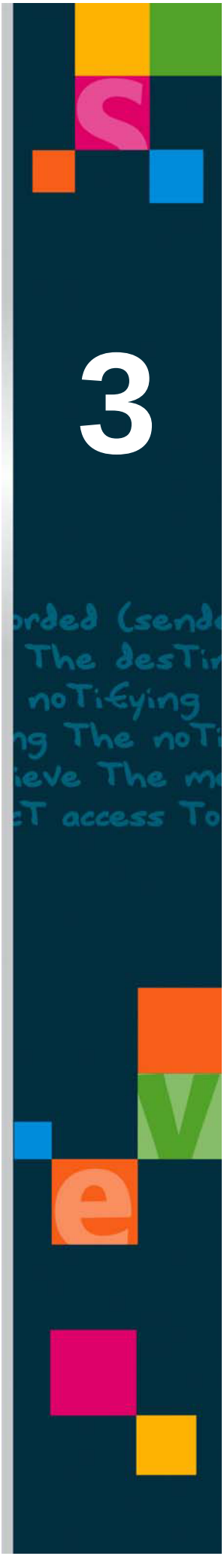
Category	Property	Default	Description
Cryptography and Key Management	sdk.keystore.path	Current directory	Fully qualified path where the keystore will be created (to store the Security Server public key). The SDK creates a keystore file `KMC.keystore` during the login process and needs to have write permission on the application base folder. The path is configured by setting this property.
Audit Management	sdk.audit.host	localhost	IP address of a remote UPA. This property can be used by client applications to send records to a remote UPA.
Identity Management and Authentication	java.security.auth.login.config	sec_jaas.conf	Login configuration file consisting of one or more entries, each specifying which underlying authentication technology should be used.
Identity Management and Authentication	javax.net.ssl.trustStore	./conf/upsec.keystore	Property used for the trust store's location (used for SSL). This is a mandatory property and must be set.
Identity Management and Authentication	security.client.context	Client_Security	An entry in a login configuration file that the applications refer to when they instantiate a `LoginContext`.
Identity Management and Authentication	security.webservice.stack	None (If a stack other than JBoss is used, set the value accordingly. For example, set the value as `jwsdp` for the JWSDP stack.)	Web services libraries to be used for web services communications.
Credentials Management	credential.secure.mode	true	If this property is set to `true`, the credential will be transported in secure mode.

By default, the SDK maintains one socket connection per audit session. Applications can override this property to open a new connection per audit record by setting the following system property:
`sdk.audit.connection=record`

In cases of integration with the WebLogic stack, additional system properties also need to be set. See "WebLogic Stack" on page 9.

Chapter 3

Security SDK Client API Reference



Identity Management API

The Identity Management API provides interfaces for the following functions:

- Performing authentication of a subject (that is, a user) during the application login process.
- Logging out a subject.
- Allowing a subject to change his or her password.
- Adding, modifying, and deleting user attributes.

The Identity Management service hosted on the Security Server exposes a web services interface for authentication of subjects. By default, the Security SDK provides a customized authentication module similar to JAAS. Instead of using the standard JAAS `LoginContext`, the client application will use the custom `LoginContext` provided by the Security SDK API. For client applications, the custom `LoginContext` is the authentication entry point. The Security SDK API handles authentication requests/responses for client applications.

The following steps summarize at a high level what happens during authentication:

1. The subject (user) submits a username and password at login.

The client application must provide some mechanism to identify the realm (that is, the security realm in the Security Server) in which the application's users are provisioned. For example, a client-side application-specific configuration file might be used for this purpose.

2. The client application instantiates the custom `SecurityServerCallbackHandler` and custom `LoginContext` provided by the Security SDK API.
3. The subject's username, password, and realm are passed to the `LoginContext` by the custom `SecurityServerCallbackHandler`.
4. The client application sets the username, password, and realm in the `LoginContext`.
5. The client application invokes the `LoginContext login()` method.
6. Internally, an authentication request containing the username, password, and realm is constructed and transmitted to the Security Server as a SAML message embedded in a SOAP payload over the HTTPS/SSL transport.
7. After the submitted identity credentials are validated at the Security Server, a session is created for the user and a SAML authentication response (that is, a token) is returned to the client application. (The term "token" is used here for simplicity. The token is also referred to as a single sign-on token, SSO token, or security token.) The token contains the following:
   - User's roles
   - User's attributes (user-level attributes and attributes inherited from the realm and groups that the user belongs to)
   - Session timers (for soft timeout and hard timeout)
   - Token expiration timestamp
   - User profile (such as first name, last name, department, phone, extension, email address, and so on)
   - Password expiry flag (indicates whether the password is about to expire)
   - Password expiry days (indicates the number of days remaining until password expiration)

- Force change password flag (indicates whether the user will be forced to change his/her password at the next login)

The client application then uses the token for session control and authorization purposes within the application. If a password is about to expire, it is the responsibility of the client application to report back to the user as appropriate, via a pop-up window or other mechanism, and allow the user the opportunity to change the password at this point.

The following sections provide information about various interfaces in the Identity Management API.

Also see "Identity Management API Code Examples" on page 91.

LoginContext

The `LoginContext` class (see Figure 1) is the primary class used for authenticating subjects (that is, users).

Figure 1 LoginContext

com.comverse.rtbd.security.sdk.idm

Class LoginContext

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.idm.LoginContext
```

public class **LoginContext**
extends java.lang.Object

Author:
snehalatha

Constructor Summary

`LoginContext(java.lang.String username, java.lang.String realm, java.lang.String passwd)`	Initalizes the Logincontext for a USER.

Method Summary

void	`addUserAttributes(java.lang.String token, java.util.HashMap userAttribMap)` Adds User Attributes(firstname,lastname,phone, resellerid,dealerid).
boolean	`changePassword(SSOToken tokenObj, java.lang.String oldPassword, java.lang.String newPassword)` Changes the User Password using token object
boolean	`changePassword(java.lang.String token, java.lang.String oldPassword, java.lang.String newPassword)` Changes the User Password using token string
void	`changeUserAttributes(java.lang.String token, java.util.HashMap userAttribMap)` Changes the User Attributes(firstname,lastname,phone, resellerid,dealerid).
ErrorCode	`getErrorCode()` Returns the ErrorCode for a given SessionID.
java.lang.String	`getSSOToken()` Returns the SSOToken string
java.lang.String	`getSSOToken(java.lang.String sessionID)` Returns the SSOToken string for a given SessionID.
javax.security.auth.Subject	`getSubject()` Returns the authenticated Subject
SSOToken	`getToken()` Convenience method for retrieving SSOToken
static java.lang.Object	`getTokenObject(java.lang.String token)` Returns the SSOToken object.
void	`login()` Perform the authentication, and if successful, associate Principals and Credentials with the authenticated Subject.
void	`logout()` Logs out the Subject
void	`relogin()` Performs re-authentication.
void	`removeUserAttributes(java.lang.String token, java.util.HashMap userAttribMap)` Removes the User Attributes(firstname,lastname,phone, resellerid,dealerid).

SSOToken

The `SSOToken` object holds authentication and attribute assertions for a user. The `SSOToken` wrapper class can be used to obtain information about session timers, attributes, roles, and other information associated with the user.

Figure 2 SSOToken

com.comverse.rtbd.security.sdk.idm.saml

Interface SSOToken

All Superinterfaces:
java.io.Serializable

All Known Implementing Classes:
SSOTokenImpl

---

public interface **SSOToken**
extends java.io.Serializable

SSOToken object holds Authentication and Attribute Assertions for User.

Method Summary

java.util.Date	getAuthInstant() Returns the time when authentication has taken place.
java.lang.String	getAuthInstantAsString() Returns the time when authentication has taken place in current Locale .
java.lang.String	getAuthUser() Returns the user being considered for authentication
boolean	getForceChangePasswordFlag() returns the Force Change Password flag
int	getPasswordExpiryDays() returns PasswordExpiryDays
boolean	getPasswordExpiryFlag() Returns the PasswordExpiryFlag
java.util.List	getRoles() Returns the ROLES associated with the user
java.lang.String[]	getRolesAsArray() Returns the ROLES associated with the user
java.lang.String	getSessionId() Returns sessionId associated with the user
SessionPolicy	getSessionTimeouts() Returns various session values - Hard Timeout (session timed out),soft Timeout(inactivity time out) etc
java.util.Date	getTokenExpiryTime() Returns token Expiry Time
java.lang.String	getTokenExpiryTimeAsString() Returns token Expiry Time in current Locale
java.util.HashMap	getUserAttributes() Returns the user attributes
UserProfile	getUserDetails() Returns user details like firstname,lastname,phonenumber,department,Email etc
java.lang.String	getUserRealm() Returns the realm of the user being considered for authentication
boolean	isValid() Checks whether the token is valid or not.
java.lang.String	toString()

SamlUtils

The `SAMLUtils` class (`com.comverse.rtbd.security.sdk.idm.saml.SAMLUtils`) contains a `getToken` utility method (see Figure 3) for deserializing the `SSOToken` object. This is useful when passing the token from one application to another.

Figure 3 SamlUtils Method Summary

Method Summary	
static java.lang.String	decrypt(java.lang.String encryptedToken) Decrypts token using password based cipher.
static java.lang.String	encrypt(java.lang.String token) Encryptes token using password based encryption
static org.opensaml.SAMLRequest	getSAMLAttributeRequest(java.lang.String user, java.lang.String realm) Generates the SAMLAttribute Request for specified user
static java.lang.String	getSAMLAuthNRequest(java.lang.String user, java.lang.String realm, java.lang.String password) Generates the SAMLAuthNRequest for specified user
static org.opensaml.SAMLRequest	getSAMLRequest(java.lang.String request) Returns deserialized SAMLRequest Object
static org.opensaml.SAMLResponse	getSAMLResponse(java.lang.String response) Returns deserialized SAMLResponse Object
static java.lang.Object	getToken(java.lang.String token) Returns deserialized token object
static void	signAssertions(org.opensaml.SAMLAssertion[] assertions, java.security.PrivateKey privateKey) Signs the Assertions
static void	signResponse(org.opensaml.SAMLResponse response, java.security.PrivateKey privateKey) Signs the SAMLResponse
static boolean	validateSignature(org.opensaml.SAMLSignedObject response, java.security.Key publicKey) Validates SAMLResponse Signatures

In order to properly use the authentication features of the Security SDK API, you must understand the basics of JAAS and how to set up the proper configuration files.

Policy Management API

The Policy Management API provides (1) a Policy Decision Point (PDP) client interface that allows client applications to perform authorization requests and (2) the ability to instantiate a PDP instance for the purpose of running a PDP in the client application (that is, an embedded PDP).

The Policy Management API provides support for two types of clients: (1) a remote PDP client that connects and exchanges XACML -related messages with the PDP services hosted on the local Unified Platform Agent, and (2) an embedded PDP client that embeds the PDP functionality in the application itself, thus removing the need for the application to connect with the UPA. The remote PDP client is preferred because it uses less memory and also separates the PDP implementation details from the client application. If the remote PDP client does not comply with the application's performance requirements, then the recommendation is to use the embedded PDP client instead of the remote client.

The following sections provide information about various interfaces in the Policy Management API. Also see "Policy Management API Code Example" on page 96.

PDP Client

Information on the `PDPClient` interface is shown in Figure 4, Figure 5 on page 21, Figure 6 on page 22, and Figure 7 on page 23.

Figure 4 PDPClient, Part 1

com.comverse.rtbd.security.sdk.policy.client

Interface PDPClient

All Known Implementing Classes:
EmbeddedPDPClient, RemotePDPClient

public interface **PDPClient**

Method Summary

int	`isAuthorized(SSOToken token, java.lang.String resource, java.lang.String action)` Method used to perform an authorization request to a PDP Accepts a deserialized SSOToken, resource (i.e.
int	`isAuthorized(java.lang.String[] subject, java.lang.String resource, java.lang.String action)` Method used to perform an authorization request to a PDP Accepts an array of subjects (i.e.
int	`isAuthorized(java.lang.String realmId, java.lang.String[] subject, java.lang.String resource, java.lang.String action)` Convenience method used to perform an authorization request to a PDP Accepts realmId, array of subjects (i.e.
int	`isAuthorized(java.lang.String serializedSSOToken, java.lang.String resource, java.lang.String action)` Method used to perform an authorization request to a PDP Accepts a serialized SSOToken, resource (i.e.
int	`reloadPolicy()` This method should be used only by embedded PDP clients to reload policies based on the policy expression supplied during the creation of the embedded PDP client.
int	`reloadPolicy(java.lang.String pExpr)` This method should be used only by embedded PDP clients to reload policies based on a policy expression.

Figure 5 PDPClient, Part 2

Method Detail

isAuthorized

```
public int isAuthorized(java.lang.String serializedSSOToken,
                        java.lang.String resource,
                        java.lang.String action)
```

Method used to perform an authorization request to a PDP Accepts a serialized SSOToken, resource (i.e. method), action (i.e. invoke)

Parameters:

resource -

action -

Returns:

authorization decision PERMIT = 0 DENY = 1 Indeterminate = 2 NotApplicable = 3

---

isAuthorized

```
public int isAuthorized(SSOToken token,
                        java.lang.String resource,
                        java.lang.String action)
```

Method used to perform an authorization request to a PDP Accepts a deserialized SSOToken, resource (i.e. method), action (i.e. invoke)

Parameters:

token -

resource -

action -

Returns:

authorization decision PERMIT = 0 DENY = 1 Indeterminate = 2 NotApplicable = 3

Figure 6 PDPClient, Part 3

isAuthorized

```
public int isAuthorized(java.lang.String[] subject,
                        java.lang.String resource,
                        java.lang.String action)
```

Method used to perform an authorization request to a PDP Accepts an array of subjects (i.e. roles), a resource, and an action

Parameters:

subject -

resource -

action -

Returns:

authorization decision. PERMIT = 0 DENY = 1 Indeterminate = 2 NotApplicable = 3

---

isAuthorized

```
public int isAuthorized(java.lang.String realmId,
                        java.lang.String[] subject,
                        java.lang.String resource,
                        java.lang.String action)
```

Convenience method used to perform an authorization request to a PDP Accepts realmId, array of subjects (i.e. roles), a resource, and an action

Parameters:

realmId -

resource -

action -

Returns:

authorization decision, PERMIT = 0 DENY = 1 INDETERMINATE = 2 NOTAPPLICABLE = 3

Figure 7 PDPClient, Part 4

reloadPolicy

```
public int reloadPolicy()
                  throws PolicyException
```

This method should be used only by embedded PDP clients to reload policies based on the policy expression supplied during the creation of the embedded PDP client. Remote PDP clients are used within the UPA agents and the functionality to resync policies already exists in the agent.

Returns:

number of policies resynced.

Throws:

PolicyException

---

reloadPolicy

```
public int reloadPolicy(java.lang.String pExpr)
                  throws PolicyException
```

This method should be used only by embedded PDP clients to reload policies based on a policy expression. Remote PDP clients are used within the UPA agents and the functionality to resync policies already exists in the agent.

Returns:

number of policies resynced

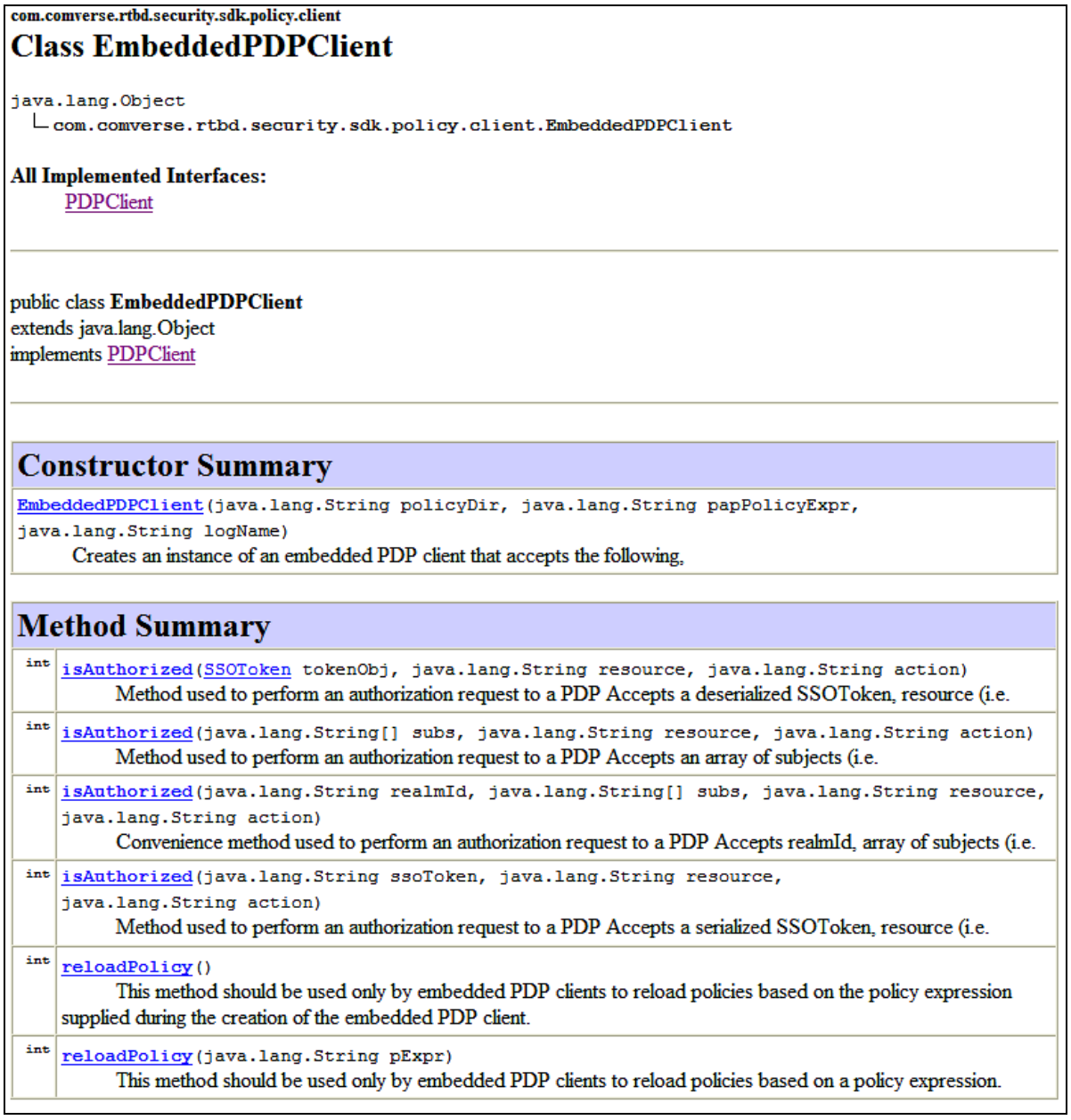
Throws:

PolicyException

Embedded PDP Client

Information on the `EmbeddedPDPClient` class is shown in Figure 8 and Figure 9 on page 25.

Figure 8 EmbeddedPDPClient, Part 1



com.comverse.rtbd.security.sdk.policy.client

Class EmbeddedPDPClient

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.policy.client.EmbeddedPDPClient
```

All Implemented Interfaces:

PDPClient

---

public class **EmbeddedPDPClient**
extends java.lang.Object
implements PDPClient

Constructor Summary

	EmbeddedPDPClient(java.lang.String policyDir, java.lang.String papPolicyExpr, java.lang.String logName) Creates an instance of an embedded PDP client that accepts the following.

Method Summary

int	isAuthorized(SSOToken tokenObj, java.lang.String resource, java.lang.String action) Method used to perform an authorization request to a PDP Accepts a deserialized SSOToken, resource (i.e.
int	isAuthorized(java.lang.String[] subs, java.lang.String resource, java.lang.String action) Method used to perform an authorization request to a PDP Accepts an array of subjects (i.e.
int	isAuthorized(java.lang.String realmId, java.lang.String[] subs, java.lang.String resource, java.lang.String action) Convenience method used to perform an authorization request to a PDP Accepts realmId, array of subjects (i.e.
int	isAuthorized(java.lang.String ssoToken, java.lang.String resource, java.lang.String action) Method used to perform an authorization request to a PDP Accepts a serialized SSOToken, resource (i.e.
int	reloadPolicy() This method should be used only by embedded PDP clients to reload policies based on the policy expression supplied during the creation of the embedded PDP client.
int	reloadPolicy(java.lang.String pExpr) This method should be used only by embedded PDP clients to reload policies based on a policy expression.

Figure 9 EmbeddedPDPClient, Part 2

Constructor Detail

EmbeddedPDPClient

```
public EmbeddedPDPClient(java.lang.String policyDir,
                         java.lang.String papPolicyExpr,
                         java.lang.String logName)
```

Creates an instance of an embedded PDP client that accepts the following,

Parameters:

`policyDir` - - directory where to store policies collected from PAP. The same folder is used to load policies during initialization.

`papPolicyExpr` - - The search expression used to get policies from PAP.

`logName` - - Used to store log statements. Optional and could be null.

Remote PDP Client

Information on the `RemotePDPClient` class is shown in Figure 10 and Figure 11 on page 27.

Figure 10 RemotePDPClient, Part 1

com.comverse.rtbd.security.sdk.policy.client

Class RemotePDPClient

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.policy.client.RemotePDPClient
```

All Implemented Interfaces:

PDPClient

---

public class **RemotePDPClient**
extends java.lang.Object
implements PDPClient

Constructor Summary

`RemotePDPClient(java.net.InetAddress addr, java.lang.String logName)`	Creates an instance of a PDP client that connects to a remote PDP service identified by the IP address.
`RemotePDPClient(java.lang.String logName)`	Creates an instance of a PDP client that connects to a PDP service on the same machine as the client (localhost).
`RemotePDPClient(java.lang.String hostname, java.lang.String logName)`	Creates an instance of a PDP client that connects to a remote PDP service identified by the hostname.

Method Summary

int	`isAuthorized(SSOToken tokenObj, java.lang.String resource, java.lang.String action)` Method used to perform an authorization request to a PDP Accepts a deserialized SSOToken, resource (i.e.
int	`isAuthorized(java.lang.String[] subs, java.lang.String resource, java.lang.String action)` Method used to perform an authorization request to a PDP Accepts an array of subjects (i.e.
int	`isAuthorized(java.lang.String realmId, java.lang.String[] subs, java.lang.String resource, java.lang.String action)` Convenience method used to perform an authorization request to a PDP Accepts realmId, array of subjects (i.e.
int	`isAuthorized(java.lang.String ssoToken, java.lang.String resource, java.lang.String action)` Method used to perform an authorization request to a PDP Accepts a serialized SSOToken, resource (i.e.
int	`reloadPolicy()` This method should be used only by embedded PDP clients to reload policies based on the policy expression supplied during the creation of the embedded PDP client.
int	`reloadPolicy(java.lang.String expr)` This method should be used only by embedded PDP clients to reload policies based on a policy expression.

Figure 11 RemotePDPClient, Part 2

Constructor Detail

RemotePDPClient

```
public RemotePDPClient(java.lang.String hostname,
                       java.lang.String logName)
                throws java.net.UnknownHostException
```

Creates an instance of a PDP client that connects to a remote PDP service identified by the hostname. Log name is optional and will be used to log information if supplied.

Parameters:

hostname -

logName -

Throws:

java.net.UnknownHostException

RemotePDPClient

```
public RemotePDPClient(java.net.InetAddress addr,
                       java.lang.String logName)
```

Creates an instance of a PDP client that connects to a remote PDP service identified by the IP address. Log name is optional and will be used to log information if supplied.

Parameters:

addr -

logName -

RemotePDPClient

```
public RemotePDPClient(java.lang.String logName)
                throws java.net.UnknownHostException
```

Creates an instance of a PDP client that connects to a PDP service on the same machine as the client (localhost). Log name is optional and will be used to log information if supplied.

Parameters:

logName -

Throws:

java.net.UnknownHostException

Audit Management API

The Audit Management API provides interfaces for creating, modifying, and publishing audit records. This API is responsible for creating audit records that conform to the XDAS specification and for the secure transmission of these records to the Unified Platform Agent.

The following sections provide information about various interfaces in the Audit Management API. Also see "Audit Management API Code Example" on page 98.

XDasSession

The `XDasSession` class (see Figure 12) allows a client application to initiate an XDAS audit session. Typically a client application establishes one session (that is, it creates only one instance of the `XDasSession` class).

Figure 12 XDasSession

com.comverse.rtbd.security.sdk.audit.xdas

Class XDasSession

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.audit.xdas.XDasSession
```

public class **XDasSession**
extends java.lang.Object

The XDasSession class allows a caller to initiate a session with the XDAS before they can use any of the services it provides. The initialization of the session supports the mutual authentication of the audit client and audit service components and establishes the audit client's XDAS authorities.

Constructor Summary

`XDasSession(java.lang.String applicationName)`
Constructor.
`XDasSession(java.lang.String sOriginatorLocationName, java.lang.String sOriginatorLocationAddress, java.lang.String sOriginatorServiceType, java.lang.String sOriginatorAuthAuthority, java.lang.String sOriginatorPrincipalName, java.lang.String sOriginatorPrincipalIdentity)`

Method Summary

java.lang.String	`getXdasHost()`
int	`getXdasPort()`
XDasRecord	`XDasStartRecord(int iEventNumber, int iOutcome, java.lang.String sInitiatorInfo, java.lang.String sTargetInfo, java.lang.String sEventInfo)` Starts a new XDAS audit record.

In the `XDasSession` class, the `XDasStartRecord` method initializes and starts a new XDAS audit record. For details on this method, see Figure 13.

Figure 13 XDasStartRecord Method Detail

Method Detail

XDasStartRecord

```
public XDasRecord XDasStartRecord(int iEventNumber,
                                  int iOutcome,
                                  java.lang.String sInitiatorInfo,
                                  java.lang.String sTargetInfo,
                                  java.lang.String sEventInfo)
                           throws XDasException
```

Starts a new XDAS audit record.

Each parameter may be "unspecified". If parameters are not specified, the audit record is initialized, but before the record can be committed, the unspecified parameters must be populated by subsequent calls to `putEventInfo`, `setEventNumber`, `setOutcome`, `setInitiatorInfo`, `setTargetInfo`, or `setEventInfo`.

Parameters:

`iEventNumber` - The event number of the detected event. Only event numbers configured as registered by the implementation will be valid. Any other event number will result in the return of an `XDasException` where the major status code is set to XDasMajorStatusCode.XDAS_S_INVALID_EVENT_NO. Use zero for unspecified.

`iOutcome` - The outcome of the event determined by the caller. Only the outcome codes listed in Table 6-7 of the XDAS specification are valid. Use XDasOutcomes.XDAS_OUT_NOT_SPECIFIED for unspecified.

`sInitiatorInfo` - The information describing the initiator in the format required by the XDAS common audit format. Use null for unspecified. It is assumed that each component of the initiator information is delimited by a colon (:) character. Components that have embedded colons or percent (%) characters should have been properly escaped by the calling application.

`sTargetInfo` - Information on the target of the event in the format required by the XDAS common audit format. Use null for unspecified. It is assumed that each component of the target information is delimited by a colon (:) character. Components that have embedded colons or percent (%) characters should have been properly escaped by the calling application.

`sEventInfo` - Information specific to the event in comma separated, name '=' value pairs. Use null for unspecified. It is assumed that embedded colons or percent (%) characters have NOT been escaped. The constructor will escape the characters if they are present in the string.

Throws:

`{@link` - openxdas.XDasException XDasException}.

`XDasException`

XDasRecord

The `XDasRecord` class (see Figure 14 on page 30) provides methods for adding/modifying an audit record. Once the audit record is populated, the application can invoke the `commit()` method, which transmits the audit record to the Unified Platform Agent or stores it on the local node.

Figure 14 XDasRecord

com.comverse.rtbd.security.sdk.audit.xdas

Class XDasRecord

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.audit.xdas.XDasRecord
```

public class **XDasRecord**
extends java.lang.Object

The XDasRecord class allows a caller to construct an audit record.

Method Summary

void	commit() Write a completed audit record to the audit stream.
void	discard() Discard an audit record.
static java.lang.String	eventInfoHelper(java.util.Map eventInfo)
void	putEventInfo(int iEventNumber, int iOutcome, java.lang.String sInitiatorInfo, java.lang.String sTargetInfo, java.lang.String sEventInfo) Add event information to an audit record or overwrite existing information.
void	setEventInfo(java.lang.String sEventInfo) Set the event information in the audit record.
void	setEventNumber(int iEventNumber) Set the event number for an audit record.
void	setExternalId(java.lang.String externalId) Method to capture external id(like subscriber id,account number).
void	setInitiatorInfo(java.lang.String sInitiatorInfo) Set the initiator information in the audit record.
void	setInitiatorInfo(java.lang.String sAuthAuthority, java.lang.String sDomainSpecificName, java.lang.String sDomainSpecificId) Set the initiator information in the audit record.
static java.lang.String	setInitiatorInfoHelper(java.lang.String source, java.lang.String username, java.lang.String sessionid)
static java.lang.String	setInitiatorInfoHelper(java.lang.String source, java.lang.String username, java.lang.String sessionid, java.lang.String rid)
void	setOutcome(int iOutcome) Set the outcome in the audit record.
void	setTargetInfo(java.lang.String sTargetInfo) Set the target information in the audit record.
void	setTargetInfo(java.lang.String sTargetLocationName, java.lang.String sTargetLocationAddress, java.lang.String sTargetServiceType, java.lang.String sTargetAuthAuthority, java.lang.String sTargetPrincipalName, java.lang.String sTargetPrincipalIdentity) Set the target information in the audit record.
static java.lang.String	setTargetInfoHelper(java.lang.String commandTag)
static java.lang.String	setTargetInfoHelper(java.lang.String commandTag, java.lang.String rid)
void	setTimeStamp() Set a timestamp for the audit record.
java.lang.String	toString()

Notable methods in the `XDasRecord` class include the following:

- The `commit` method sends the audit record to a local or remote Unified Platform Agent. If the UPA is not available, the record will be stored locally in an audit file under the `$JBOSS_HOME/sdkaudit` directory. After a record is committed, all fields are cleared.
- The `discard` method discards an audit record (clears all the fields in the record).
- For the `setEventNumber` method, an event number can be chosen from the `XdasEvents` class.
- The `setExternald` method captures an external ID (such as subscriber ID or account number) and appends it to the `eventinfo` field in the audit record.
- For the `setOutcome` method, the `XDasOutcomes` class provides a set of valid event outcome codes.
- The `toString` method returns the audit record in XDAS format.

For details on all the methods in the `XDasRecord` class, see the following figures:

- Figure 15, "XDasRecord Method Detail, Part 1," on page 32
- Figure 16, "XDasRecord Method Detail, Part 2," on page 33
- Figure 17, "XDasRecord Method Detail, Part 3," on page 34
- Figure 18, "XDasRecord Method Detail, Part 4," on page 35
- Figure 19, "XDasRecord Method Detail, Part 5," on page 36

For information on the `XdasEvents` class (for event numbers) and the `XDasOutcomes` class (for event outcome codes), see "Audit Constants (XDasEvents and XDasOutcomes)" on page 36.

Figure 15 XDasRecord Method Detail, Part 1

Method Detail

putEventInfo

```
public void putEventInfo(int iEventNumber,
                         int iOutcome,
                         java.lang.String sInitiatorInfo,
                         java.lang.String sTargetInfo,
                         java.lang.String sEventInfo)
                  throws XDasException
```

Add event information to an audit record or overwrite existing information.

Although several parameters may be "unspecified", an application must at some point have specified all parameters before a call to `commit` may be made. Note that an "empty" parameter is different than an "unspecified" parameter.

The caller must have the XDAS_AUDIT_SUBMIT authority.

Parameters:

`iEventNumber` - The event number of the detected event. Only event numbers configured as registered by the implementation will be valid. Any other event number will result in the return of an `XDasException` where the major status code is set to XDasMajorStatusCode.XDAS_S_INVALID_EVENT_NO. Use zero for unspecified.
`iOutcome` - The outcome of the event determined by the caller. Only the outcome codes listed in Table 6-7 of the XDAS specification are valid. Use XDasOutcomes.XDAS_OUT_NOT_SPECIFIED for unspecified.
`sInitiatorInfo` - The information describing the initiator in the format required by the XDAS common audit format. Use null for unspecified. It is assumed that each component of the initiator information is delimited by a colon (:) character. Components that have embedded colons or percent (%) characters should have been properly escaped by the calling application.
`sTargetInfo` - Information on the target of the event in the format required by the XDAS common audit format. Use null for unspecified. It is assumed that each component of the target information is delimited by a colon (:) character. Components that have embedded colons or percent (%) characters should have been properly escaped by the calling application.
`sEventInfo` - Information specific to the event in comma separated, name '=' value pairs. Use null for unspecified. It is assumed that embedded colons or percent (%) characters will have been properly escaped by the calling application.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

---

setEventNumber

```
public void setEventNumber(int iEventNumber)
                    throws XDasException
```

Set the event number for an audit record.

The caller must have the XDAS_AUDIT_SUBMIT authority.

Parameters:

`iEventNumber` - The event number of the detected event. Only event numbers configured as registered by the implementation will be valid. Any other event number will result in the return of an `XDasException` where the major status code is set to XDasMajorStatusCode.XDAS_S_INVALID_EVENT_NO. Use zero for unspecified.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

---

setOutcome

```
public void setOutcome(int iOutcome)
                throws XDasException
```

Set the outcome in the audit record.

The caller must have the XDAS_AUDIT_SUBMIT authority.

Parameters:

`iOutcome` - The outcome of the event determined by the caller. Only the outcome codes listed in Table 6-7 of the XDAS specification are valid. The value of XDasOutcomes.XDAS_OUT_NOT_SPECIFIED may not be used in this method.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

Figure 16 XDasRecord Method Detail, Part 2

setTargetInfo

```
public void setTargetInfo(java.lang.String sTargetInfo)
                   throws XDasException
```

Set the target information in the audit record.

The caller must have the XDAS_AUDIT_SUBMIT authority.

Parameters:

`sTargetInfo` - Information on the target of the event in the format required by the XDAS common audit format. A value of null is not allowed in this method. It is assumed that each component of the target information is delimited by a colon (:) character. Components that have embedded colons or percent (%) characters should have been properly escaped by the calling application.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

---

setTargetInfo

```
public void setTargetInfo(java.lang.String sTargetLocationName,
                          java.lang.String sTargetLocationAddress,
                          java.lang.String sTargetServiceType,
                          java.lang.String sTargetAuthAuthority,
                          java.lang.String sTargetPrincipalName,
                          java.lang.String sTargetPrincipalIdentity)
                   throws XDasException
```

Set the target information in the audit record.

The caller must have the XDAS_AUDIT_SUBMIT authority.

It is assumed that each of the parameters passed into this method will NOT have colon (:) and percent (%) characters escaped. This method will reformat the strings to insert needed escape characters.

Parameters:

`sTargetLocationName` - The location name component of the target information.
`sTargetLocationAddress` - The location address component of the target information.
`sTargetServiceType` - The service type component of the target information.
`sTargetAuthAuthority` - The authorization authority component of the target information.
`sTargetPrincipalName` - The principal name component of the target information.
`sTargetPrincipalIdentity` - The principal identity component of the target information.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

Figure 17 XDasRecord Method Detail, Part 3

setInitiatorInfo

```
public void setInitiatorInfo(java.lang.String sInitiatorInfo)
                      throws XDasException
```

Set the initiator information in the audit record.

The caller must have the XDAS_AUDIT_SUBMIT authority.

Parameters:

`sInitiatorInfo` - The information describing the initiator in the format required by the XDAS common audit format. A value of null is not allowed in this method. It is assumed that each component of the initiator information is delimited by a colon (:) character. Components that have embedded colons or percent (%) characters should have been properly escaped by the calling application.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

---

setInitiatorInfo

```
public void setInitiatorInfo(java.lang.String sAuthAuthority,
                             java.lang.String sDomainSpecificName,
                             java.lang.String sDomainSpecificId)
                      throws XDasException
```

Set the initiator information in the audit record.

The caller must have the XDAS_AUDIT_SUBMIT authority.

It is assumed that each of the parameters passed into this method will NOT have colon (:) and percent (%) characters escaped. This method will reformat the strings to insert needed escape characters.

Parameters:

`sAuthAuthority` - The authorization authority component of the initiator information.
`sDomainSpecificName` - The domain specific name component of the initiator information.
`sDomainSpecificId` - The domain specific ID component of the initiator information.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

Figure 18 XDasRecord Method Detail, Part 4

setEventInfo

```
public void setEventInfo(java.lang.String sEventInfo)
                  throws XDasException
```

Set the event information in the audit record.

The caller must have the XDAS_AUDIT_SUBMIT authority.

Parameters:

`sEventInfo` - Information specific to the event in comma separated, name '=' value pairs. A value of null is not allowed in this method. It is assumed that this parameter does NOT have colon (:) and percent (%) characters escaped. This method will reformat the parameter to insert needed escape characters.

Throws:

`{@link` - openxdas.XDasException XDasException}.
XDasException

---

setInitiatorInfoHelper

```
public static java.lang.String setInitiatorInfoHelper(java.lang.String source,
                                                       java.lang.String username,
                                                       java.lang.String sessionid)
```

---

setInitiatorInfoHelper

```
public static java.lang.String setInitiatorInfoHelper(java.lang.String source,
                                                       java.lang.String username,
                                                       java.lang.String sessionid,
                                                       java.lang.String rid)
```

---

setTargetInfoHelper

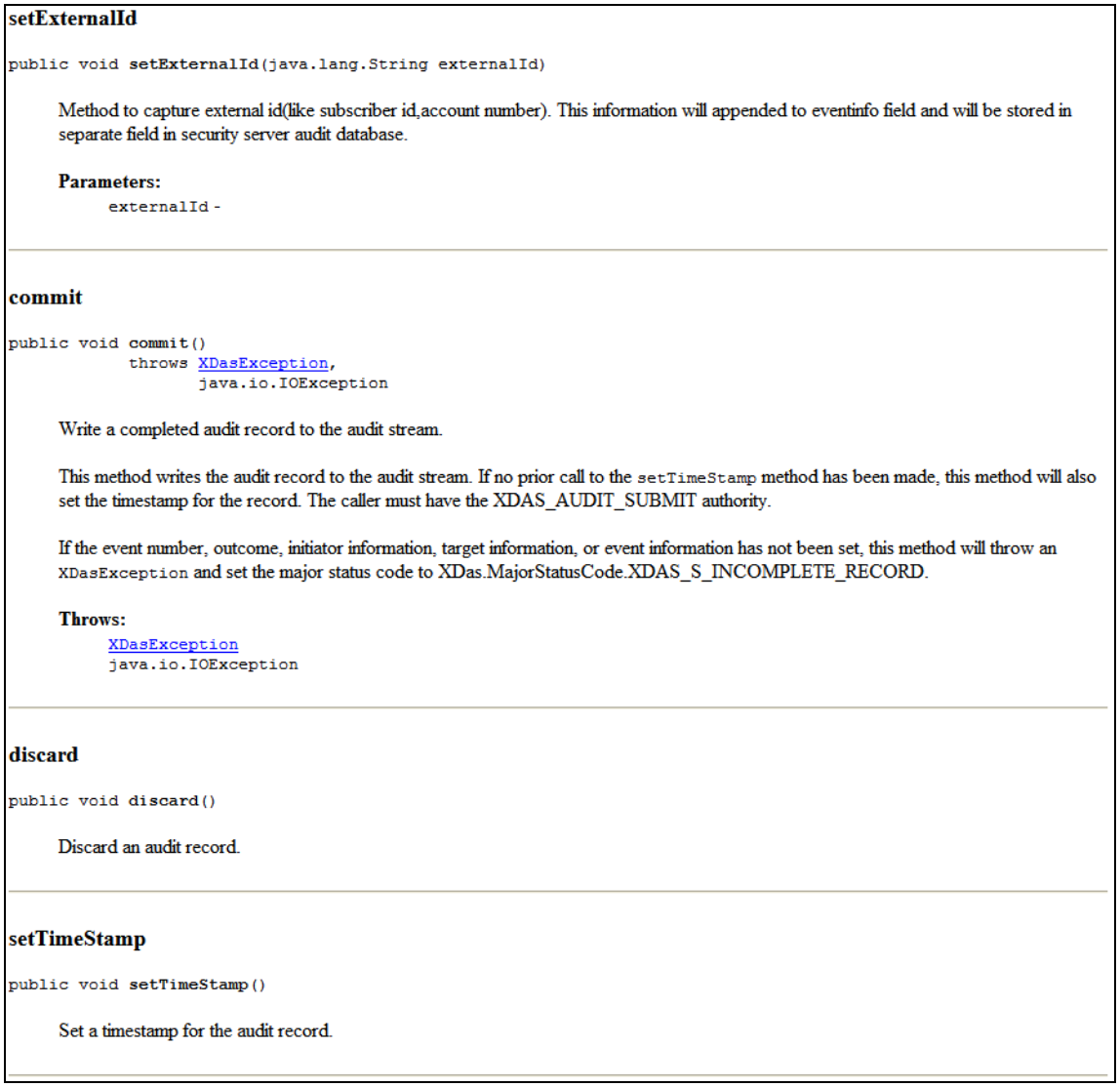
```
public static java.lang.String setTargetInfoHelper(java.lang.String commandTag)
```

---

setTargetInfoHelper

```
public static java.lang.String setTargetInfoHelper(java.lang.String commandTag,
                                                   java.lang.String rid)
```

Figure 19 XDasRecord Method Detail, Part 5



setExternalId

```
public void setExternalId(java.lang.String externalId)
```

Method to capture external id(like subscriber id,account number). This information will appended to eventinfo field and will be stored in separate field in security server audit database.

Parameters:

externalId -

---

commit

```
public void commit()
            throws XDasException,
                   java.io.IOException
```

Write a completed audit record to the audit stream.

This method writes the audit record to the audit stream. If no prior call to the `setTimeStamp` method has been made, this method will also set the timestamp for the record. The caller must have the XDAS_AUDIT_SUBMIT authority.

If the event number, outcome, initiator information, target information, or event information has not been set, this method will throw an `XDasException` and set the major status code to XDas.MajorStatusCode.XDAS_S_INCOMPLETE_RECORD.

Throws:

XDasException
java.io.IOException

---

discard

```
public void discard()
```

Discard an audit record.

---

setTimeStamp

```
public void setTimeStamp()
```

Set a timestamp for the audit record.

Audit Constants (XDasEvents and XDasOutcomes)

The `XDasEvents` class provides a set of all valid event codes, and the `XDasOutcomes` class provides a set of all valid outcome codes.

An application developer or integrator who submits or imports security-domain-specific events to the XDAS service must map those events to the XDAS generic events.

XDAS Events Constants

An application developer can use the numeric constants defined in the `XDasEvents` class (see Figure 20) to set the audit event number in `XDasRecord`. For information about the meaning of the event codes, see the "Audit Management" chapter in the *Security Platform Operations Guide*.

Figure 20 Constants in XDasEvents

com.comverse.rtbd.security.sdk.audit.xdas.XDasEvents

public static final int XDAS_AE_AUD_CONFIG	16777259
public static final int XDAS_AE_AUD_DS_CORR	16777261
public static final int XDAS_AE_AUD_DS_FULL	16777260
public static final int XDAS_AE_BACKUP_DATASTORE	16777257
public static final int XDAS_AE_CREATE_ACCOUNT	16777217
public static final int XDAS_AE_CREATE_DATA_ITEM	16777227
public static final int XDAS_AE_CREATE_DATA_ITEM_ASSOC	16777247
public static final int XDAS_AE_CREATE_PEER_ASSOC	16777241
public static final int XDAS_AE_CREATE_SESSION	16777223
public static final int XDAS_AE_DELETE_ACCOUNT	16777218
public static final int XDAS_AE_DELETE_DATA_ITEM	16777228
public static final int XDAS_AE_DISABLE_ACCOUNT	16777219
public static final int XDAS_AE_DISABLE_SERVICE	16777235
public static final int XDAS_AE_ENABLE_ACCOUNT	16777220
public static final int XDAS_AE_ENABLE_SERVICE	16777236
public static final int XDAS_AE_INSTALL_SERVICE	16777231
public static final int XDAS_AE_INVOKE_SERVICE	16777237
public static final int XDAS_AE_MODIFY_ACCOUNT	16777222
public static final int XDAS_AE_MODIFY_ASSOC_CONTEXT	16777244
public static final int XDAS_AE_MODIFY_DATA_ITEM_ASSOC_CONTEXT	16777250
public static final int XDAS_AE_MODIFY_DATA_ITEM_ATT	16777230
public static final int XDAS_AE_MODIFY_DATA_ITEM_CONTENTS	16777252
public static final int XDAS_AE_MODIFY_PROCESS_CONTEXT	16777240
public static final int XDAS_AE_MODIFY_SERVICE_CONFIG	16777234
public static final int XDAS_AE_MODIFY_SESSION	16777226
public static final int XDAS_AE_NOT_SPECIFIED	0
public static final int XDAS_AE_QUERY_ACCOUNT	16777221
public static final int XDAS_AE_QUERY_ASSOC_CONTEXT	16777243
public static final int XDAS_AE_QUERY_DATA_ITEM_ASSOC_CONTEXT	16777249
public static final int XDAS_AE_QUERY_DATA_ITEM_ATT	16777229
public static final int XDAS_AE_QUERY_DATA_ITEM_CONTENTS	16777251
public static final int XDAS_AE_QUERY_PROCESS_CONTEXT	16777239
public static final int XDAS_AE_QUERY_SERVICE_CONFIG	16777233
public static final int XDAS_AE_QUERY_SESSION	16777225
public static final int XDAS_AE_RECEIVE_DATA_VIA_ASSOC	16777245
public static final int XDAS_AE_RECOVER_DATASTORE	16777258
public static final int XDAS_AE_REMOVE_SERVICE	16777232
public static final int XDAS_AE_RESOURCE_CORRUPT	16777256
public static final int XDAS_AE_RESOURCE_EXHAUST	16777255
public static final int XDAS_AE_SEND_DATA_VIA_ASSOC	16777246
public static final int XDAS_AE_SHUTDOWN_SYS	16777254
public static final int XDAS_AE_START_SYS	16777253
public static final int XDAS_AE_TERMINATE_DATA_ITEM_ASSOC	16777248
public static final int XDAS_AE_TERMINATE_PEER_ASSOC	16777242
public static final int XDAS_AE_TERMINATE_SERVICE	16777238
public static final int XDAS_AE_TERMINATE_SESSION	16777224

XDAS Outcome Constants

The `XDasOutcomes` class provides a set of all valid XDAS outcome codes. XDAS outcome codes represent the outcome of a given event. Application developers can use the numeric constants defined in the `XDasOutcomes` class (see Figure 21) to set the audit outcome number in `XdasRecord`. For information about the meaning of outcome codes, see the "Audit Management" chapter in the *Security Platform Operations Guide*.

Figure 21 Constants in XDasOutcomes

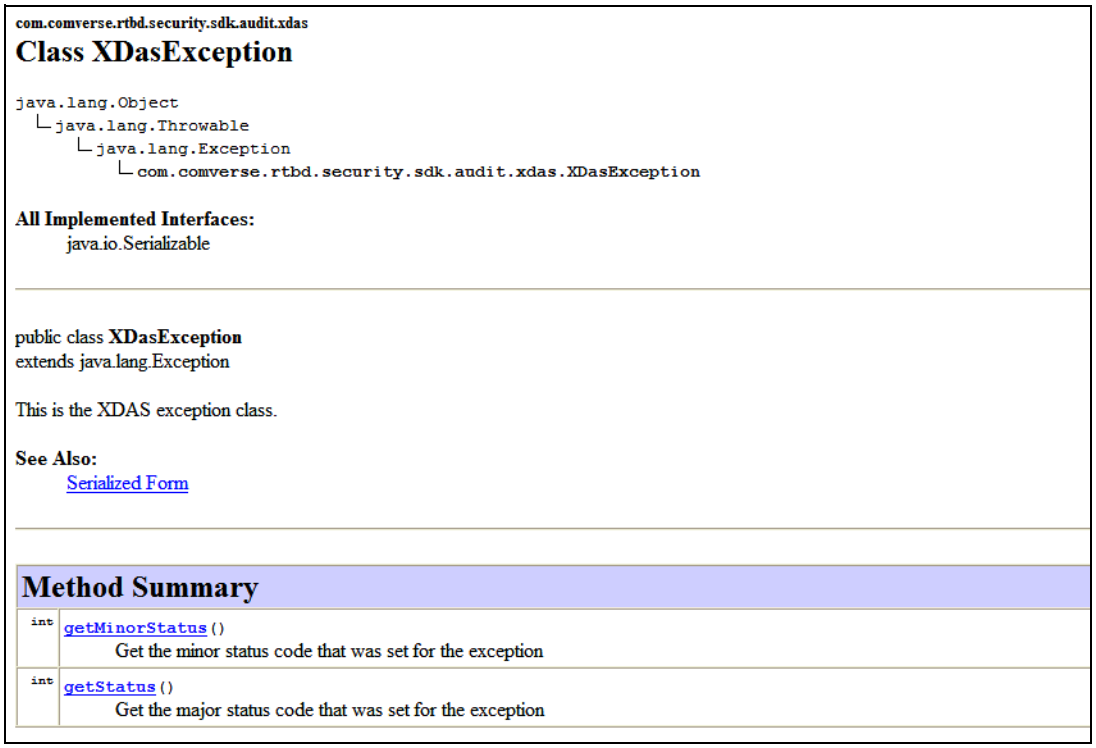
com.comverse.rtbd.security.sdk.audit.xdas.XDasOutcomes

public static final int XDAS_OUT_ACTIONS_SET	8192
public static final int XDAS_OUT_ALREADY_DISABLED	8193
public static final int XDAS_OUT_ALREADY_ENABLED	4097
public static final int XDAS_OUT_BUSY	32769
public static final int XDAS_OUT_DENIAL	2
public static final int XDAS_OUT_DISABLED	65537
public static final int XDAS_OUT_ENTITY_EXISTS	262145
public static final int XDAS_OUT_ENTITY_NON_EXISTENT	524289
public static final int XDAS_OUT_FAILURE	1
public static final int XDAS_OUT_HARDWARE_FAILURE	1025
public static final int XDAS_OUT_INSUFFICIENT_PRIVILEGE	258
public static final int XDAS_OUT_INVALID_CREDENTIALS	1026
public static final int XDAS_OUT_INVALID_IDENTITY	514
public static final int XDAS_OUT_INVALID_INPUT	131073
public static final int XDAS_OUT_LOST_ASSOCIATION	2049
public static final int XDAS_OUT_NOT_SPECIFIED	-1
public static final int XDAS_OUT_PRESELECT_CRITERIA_SET	2048
public static final int XDAS_OUT_PRIV_GRANTED	512
public static final int XDAS_OUT_PRIV_REVOKED	1024
public static final int XDAS_OUT_PRIV_USED	256
public static final int XDAS_OUT_SERVICE_ERROR	16385
public static final int XDAS_OUT_SERVICE_FAILURE	513
public static final int XDAS_OUT_SERVICE_UNAVAILABLE	257
public static final int XDAS_OUT_SUCCESS	0
public static final int XDAS_OUT_THRESHOLDS_SET	4096

XDasException

The `XDasException` class (see Figure 22 on page 39) wraps the major status code and minor status code defined in the XDAS specification.

Figure 22 XDasException



com.comverse.rtbd.security.sdk.audit.xdas

Class XDasException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └com.comverse.rtbd.security.sdk.audit.xdas.XDasException
```

All Implemented Interfaces:
java.io.Serializable

public class **XDasException**
extends java.lang.Exception

This is the XDAS exception class.

See Also:
Serialized Form

Method Summary

int	getMinorStatus() Get the minor status code that was set for the exception
int	getStatus() Get the major status code that was set for the exception

If the Audit Management API is not able to connect to the Unified Platform Agent, audit files will be stored locally under the `$JBOSS_HOME/sdkaudit` directory.

If the `$JBOSS_HOME` environment variable is not defined, the Audit Management API will throw an `XDasException` with the major status set to 25 when storing audit records locally.

Credentials Management API

The Credentials Management API provides interfaces for fetching credentials, such as database passwords for business applications, from the Security Server database or a local password cache. Currently in the Comverse ONE solution, the supported credential types are database passwords for business databases used by applications and SNMP community strings for network devices.

The following sections provide information about various interfaces in the Credentials Management API. Also see "Credentials Management API Code Examples" on page 100.

Database Password Management

The `DBPasswordManagement` class (see Figure 23) provides methods for fetching database passwords. They can be fetched either from the Security Server database or from a local password cache.

Figure 23 DBPasswordManagement

com.comverse.rtbd.security.sdk.dbpwd

Class DBPasswordManagement

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.dbpwd.DBPasswordManagement
```

public class **DBPasswordManagement**
extends java.lang.Object

Class for fethcing DB Password from security server or from Local cache.

Constructor Summary

DBPasswordManagement()

Method Summary

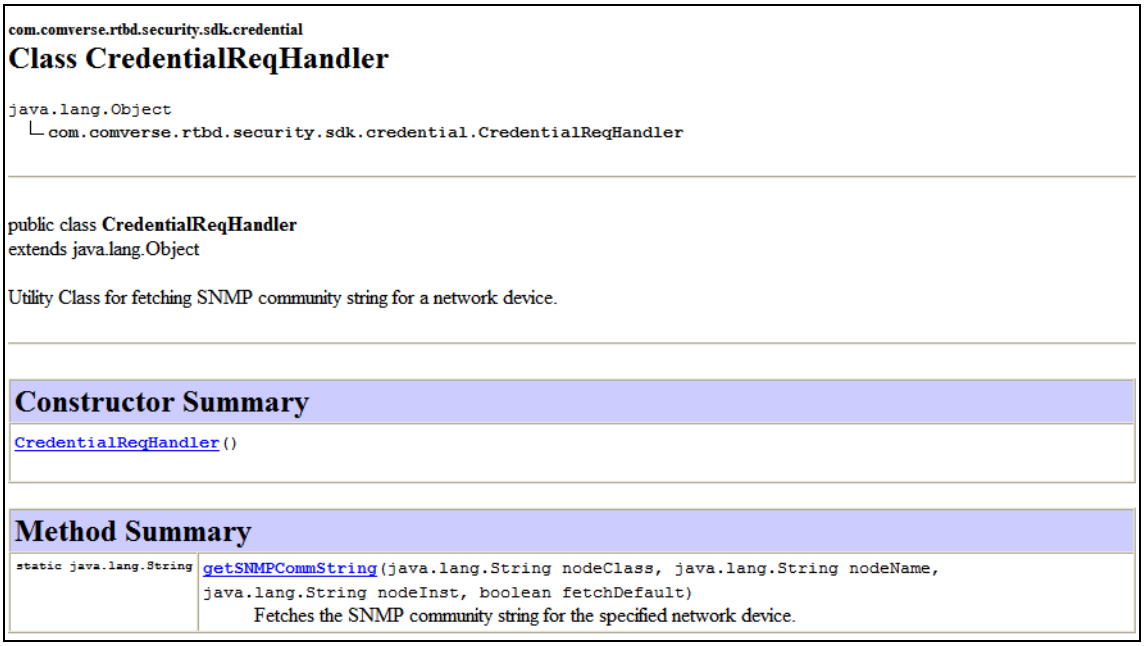
static java.lang.String	getDBPassword(java.lang.String dbType, java.lang.String instance, java.lang.String uName) Fetches the password from local cache or from security server.First the local cache will be searched for password. If not present in local cache then a request is sent to security server for fetching password and local cache is updated.
static java.lang.String	getDBPassword(java.lang.String dbType, java.lang.String instance, java.lang.String uName, boolean fetchLocally) Fetches the password from local cache or from security server.

The API will create the DBPassword cache under the `$JBOSS_HOME/dbpwd` directory. If the `$JBOSS_HOME` environment variable is not defined, the cache will be created under the `/tmp` directory.

SNMP Community Strings

The `CredentialReqHandler` class (see Figure 24 on page 41) provides a method for fetching SNMP community strings from the Security Server database.

Figure 24 CredentialReqHandler



com.comverse.rtbd.security.sdk.credential

Class CredentialReqHandler

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.credential.CredentialReqHandler
```

public class **CredentialReqHandler**
extends java.lang.Object

Utility Class for fetching SNMP community string for a network device.

Constructor Summary

CredentialReqHandler()

Method Summary

static java.lang.String	getSNMPCommString(java.lang.String nodeClass, java.lang.String nodeName, java.lang.String nodeInst, boolean fetchDefault) Fetches the SNMP community string for the specified network device.

Key Management API

The Key Management API facilitates centralized management of all symmetric keys used for encryption. The Key Management API exposes interfaces for creating, fetching, and disabling symmetric keys. It also provides interfaces for fetching the Security Server public key and creating asymmetric key pairs.

Only symmetric keys are stored in the Security Server database. Asymmetric keys are not stored in the database.

The following sections provide information about various interfaces in the Key Management API. Also see "Key Management API Code Example" on page 102.

KeyManagerClient

The `KeyManagerClient` class (see Figure 25 on page 42) serves as the primary interface between a client application and the Key Management server.

Figure 25 KeyManagerClient

com.comverse.rtbd.security.sdk.key

Class KeyManagerClient

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.key.KeyManagerClient
```

public final class **KeyManagerClient**
extends java.lang.Object

This class serves as primary interface between client application and key managemnt server.

Method Summary

boolean	disableKey(java.lang.String gkid) Changes the status of the key from active to inactive. Returns true on suceess or false on failure.
java.security.KeyPair	generateDSAKeyPair() Returns a DSA KeyPair Object.
SymKey	generateKey(java.lang.String algo) Creates and stores a new symmetric key object for specified algorithm. Returns Symkey object.
SymKey	generateKey(java.lang.String algo, java.lang.String kid) Creates and stores a new symmetric key object for specified algorithm and assoisates it with user defined key Id.
java.security.KeyPair	generateRSAKeyPair() Returns a RSA KeyPair Object.
javax.crypto.SecretKey	genereateRandomAESKey() Method to genarate AES secret Key.
static KeyManagerClient	getInstance() returns an instance of KeyManagerClient class
SymKey	getKey(java.lang.String gkid) Fetches the symmetric key referd by global key ID from security server. Retuns SymKey object.
java.security.PrivateKey	getSecurityServerPrivateKey() Only for testing purpose...
java.security.PublicKey	getSecurityServerPublicKey() Fetches security server public key

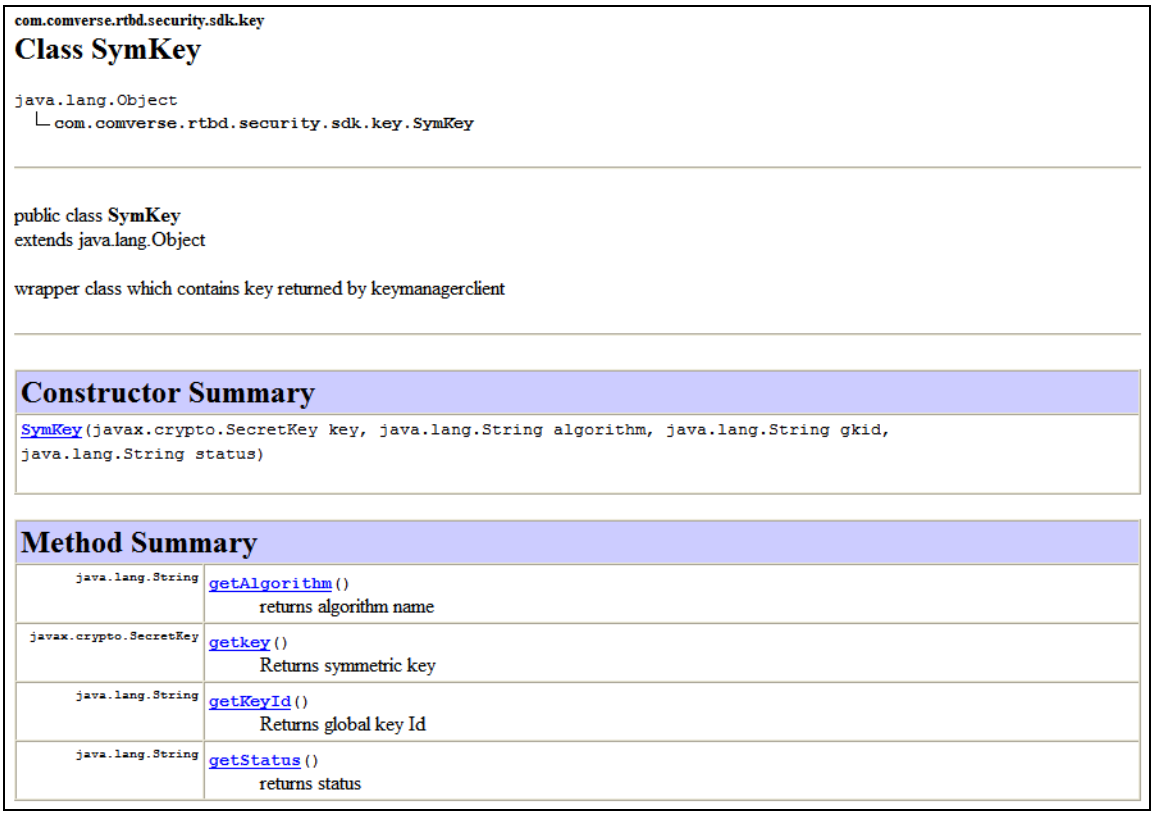
NOTE By default, `KeyManagerClient` creates a keystore for storing the Security Server public key in the current working directory. Applications can override this default by using the `sdk.keystore.path` system property.

NOTE The length of the symmetric key is configurable only in the Security Server. Client applications cannot configure the length of the key to be generated.

SymKey

The `SymKey` class (see Figure 26) creates a wrapper object that contains the key ID, key algorithm, key value, and key status for symmetric keys.

Figure 26 SymKey



com.comverse.rtbd.security.sdk.key

Class SymKey

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.key.SymKey
```

public class **SymKey**
extends java.lang.Object

wrapper class which contains key returned by keymanagerclient

Constructor Summary

SymKey(javax.crypto.SecretKey key, java.lang.String algorithm, java.lang.String gkid, java.lang.String status)

Method Summary

java.lang.String	getAlgorithm() returns algorithm name
javax.crypto.SecretKey	getkey() Returns symmetric key
java.lang.String	getKeyId() Returns global key Id
java.lang.String	getStatus() returns status

Cryptography API

The Cryptography API provides classes for symmetric encryption, asymmetric encryption, and password-based encryption. It also provides classes for digitally signing/verifying signed data and for creating and verifying message digests.

The following sections provide information about various interfaces in the Cryptography API. Also see "Cryptography API Code Examples" on page 104.

Symmetric/Asymmetric Encryption

The `EncryptorProvider` class (see Figure 27) serves as a factory class for creating instances of symmetric encryptors and asymmetric encryptors.

Currently, the Security SDK API supports the following two symmetric algorithms for encryption: AES and Blowfish.

Figure 27 EncryptorProvider

com.comverse.rtbd.security.sdk.crypto

Class EncryptorProvider

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.EncryptorProvider
```

public class **EncryptorProvider**
extends java.lang.Object

Engine classes to get an instance of symmetricencryptor or asymmetricencryptor

Constructor Summary

EncryptorProvider()

Method Summary

static SymmetricEncryptor	createAESEncryptor()
static SymmetricEncryptor	createBlowfishEncryptor()
static AsymmetricEncryptor	createRSAEncryptor()

Symmetric Encryptors

The `AESEncryptor` class and `BlowfishEncryptor` class provide interfaces for encryption/decryption of different data types such as strings, bytes, Objects, and files.

The `AESEncryptor` class (see Figure 28) uses the AES algorithm to perform encryption and decryption operations.

Figure 28 AESEncryptor

com.comverse.rtbd.security.sdk.crypto.encryptors

Class AESEncryptor

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.encryptors.AESEncryptor
```

All Implemented Interfaces:
SymmetricEncryptor

public class **AESEncryptor**
extends java.lang.Object
implements SymmetricEncryptor

Class to encrypt/decrypt data using AES algorithm.Supports encryption/decryption of byte array,string and serializable object.

Constructor Summary

AESEncryptor()	Creates a new instance of AESEncryptor

Method Summary

byte[]	decrypt(byte[] in)
java.lang.Object	decrypt(javax.crypto.SealedObject so)
java.lang.String	decrypt(java.lang.String str)
void	decrypt(java.lang.String inputFile, java.lang.String outputFile) Interface for decrypting a file.
byte[]	encrypt(byte[] in)
javax.crypto.SealedObject	encrypt(java.io.Serializable ob)
java.lang.String	encrypt(java.lang.String str)
void	encrypt(java.lang.String inputFile, java.lang.String outputFile) Interface for encrypting a file.
javax.crypto.SecretKey	getKey() Returns SecretKey
void	setKey(javax.crypto.SecretKey key) Intializes secretKey

The `BlowfishEncryptor` class (see Figure 29 on page 46) uses the Blowfish algorithm to perform encryption/decryption operations.

Figure 29 BlowfishEncryptor

com.comverse.rtbd.security.sdk.crypto.encryptors

Class BlowfishEncryptor

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.encryptors.BlowfishEncryptor
```

All Implemented Interfaces:
SymmetricEncryptor

public class **BlowfishEncryptor**
extends java.lang.Object
implements SymmetricEncryptor

Class to encrypt/decrypt data using Blowfish algorithm.Supports encryption/decryption of byte array,string and serializable object

Constructor Summary

BlowfishEncryptor()	Creates a new instance of BlowfishEncryptor

Method Summary

byte[]	decrypt(byte[] in)
java.lang.Object	decrypt(javax.crypto.SealedObject so)
java.lang.String	decrypt(java.lang.String str)
void	decrypt(java.lang.String inputFile, java.lang.String outputFile) Interface for decrypting a file.
byte[]	encrypt(byte[] in)
javax.crypto.SealedObject	encrypt(java.io.Serializable ob)
java.lang.String	encrypt(java.lang.String str)
void	encrypt(java.lang.String inputFile, java.lang.String outputFile) Interface for encrypting a file.
javax.crypto.SecretKey	getKey() Returns SecretKey
void	setKey(javax.crypto.SecretKey key) Intializes secretKey

For the `encrypt` and `decrypt` methods that take `inputFile` and `outputFile` as parameters, the input file and output file must be different.

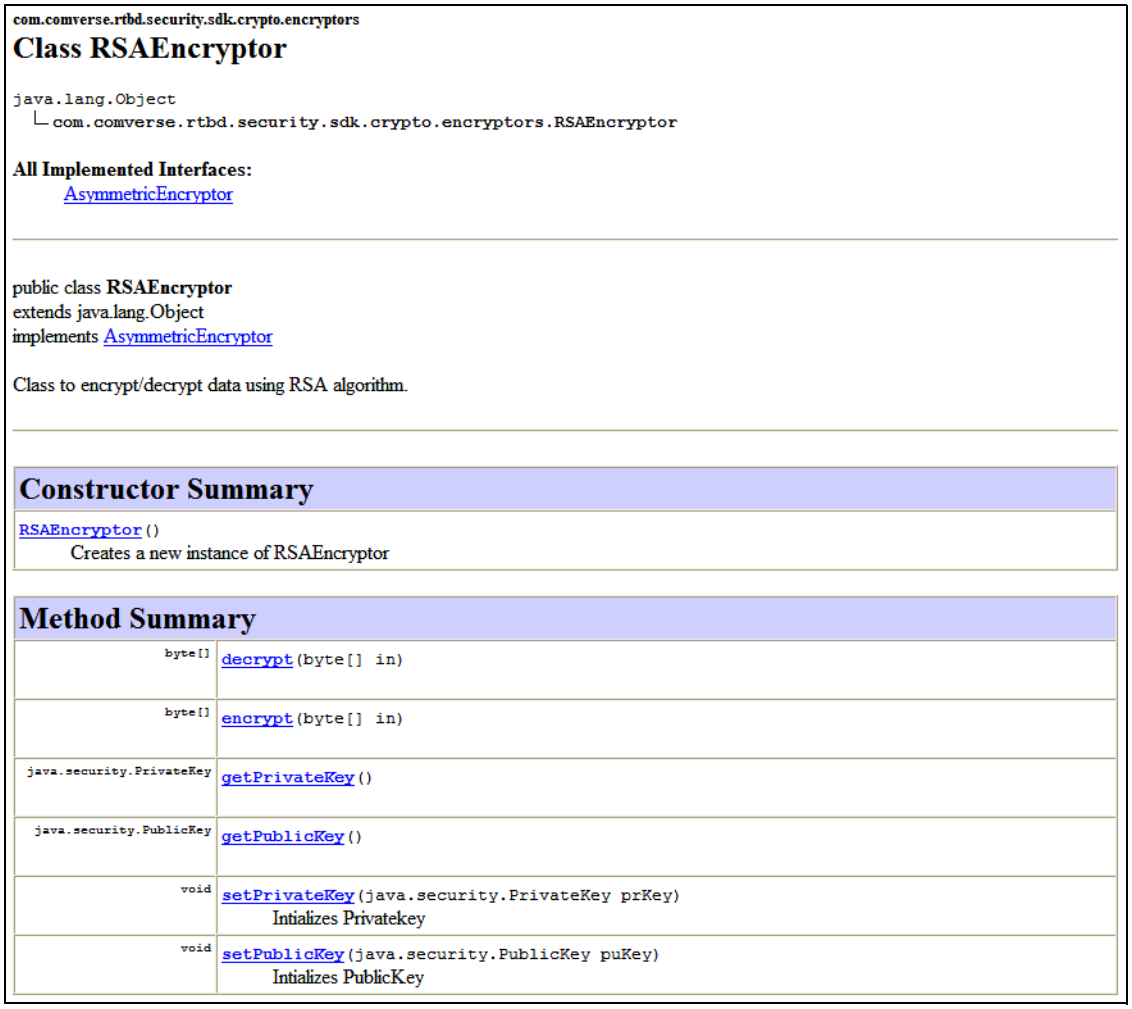
The Security SDK API uses Cipher Block Chaining (CBC) for all symmetric encryptions and the PKCS#5 padding scheme for padding the last block. These parameters are not configurable.

If the key size used for encryption is greater than 128 bits, client applications should install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.

Asymmetric Encryptor

The `RSAEncryptor` class (see Figure 30) provides interfaces for encryption/decryption of byte arrays using the RSA algorithm.

Figure 30 RSAEncryptor



com.comverse.rtbd.security.sdk.crypto.encryptors

Class RSAEncryptor

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.encryptors.RSAEncryptor
```

All Implemented Interfaces:
AsymmetricEncryptor

public class **RSAEncryptor**
extends java.lang.Object
implements AsymmetricEncryptor

Class to encrypt/decrypt data using RSA algorithm.

Constructor Summary

RSAEncryptor()	Creates a new instance of RSAEncryptor

Method Summary

byte[]	decrypt(byte[] in)
byte[]	encrypt(byte[] in)
java.security.PrivateKey	getPrivateKey()
java.security.PublicKey	getPublicKey()
void	setPrivateKey(java.security.PrivateKey prKey) Intializes Privatekey
void	setPublicKey(java.security.PublicKey puKey) Intializes PublicKey

Although in principle it is possible to encrypt data using either the private key or public key, the Security SDK API allows encryption of data using the public key only. That is, `RSAEncryptor` uses the public key to perform the encryption operation and the private key to perform the decryption operation.

Password-Based Encryption

The password-based encryption (PBE) algorithm uses passwords for encryption and decryption of data (strings or bytes) instead of secret keys.

PBE String Encryption

The `StandardPBEStringEncryptor` class (see Figure 31 and Figure 32 on page 49) provides interfaces to encrypt strings using a password.

Figure 31 StandardPBEStringEncryptor, Part 1

com.comverse.rtbd.security.sdk.crypto.pbe

Class StandardPBEStringEncryptor

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.pbe.StandardPBEStringEncryptor
```

All Implemented Interfaces:
PBEStringEncryptor, StringEncryptor

---

public final class **StandardPBEStringEncryptor**
extends java.lang.Object
implements PBEStringEncryptor

Standard implementation of the `PBEStringEncryptor` interface. This class lets the user specify the algorithm to be used for encryption, the password to use, and the number of hashing iterations that will be applied for obtaining the encryption key.

This class avoids byte-conversion problems related to the fact of different platforms having different default charsets, and returns encryption results in the form of BASE64-encoded ASCII Strings.

This class is *thread-safe*.

Configuration

The algorithm, password and key-obtention iterations can take values in any of these ways:

- Using its default values (except for password).
- Setting a `PBEConfig` object which provides new configuration values.
- Calling the corresponding `setAlgorithm`, `setPassword` or `setKeyObtentionIterations` methods.

And the actual values to be used for initialization will be established by applying the following priorities:

1. First, the default values are considered (except for password).
2. Then, if a `PBEConfig` object has been set with `setConfig`, the non-null values returned by its `getX` methods override the default values.
3. Finally, if the corresponding `setX` method has been called on the encryptor itself for any of the configuration parameters, the values set by these calls override all of the above.

Initialization

Before it is ready to encrypt, an object of this class has to be *initialized*. Initialization happens:

- When `initialize` is called.
- When `encrypt` or `decrypt` are called for the first time, if `initialize` has not been called before.

Once an encryptor has been initialized, trying to change its configuration will result in an `AlreadyInitializedException` being thrown.

Usage

An ecryptor may be used for:

- *Encrypting messages*, by calling the `encrypt` method.
- *Decrypting messages*, by calling the `decrypt` method.

Because of the use of a random salt, two encryption results for the same message will always be different (except in the case of random salt coincidence). This enforces security by difficulting brute force attacks on sets of data at a time and forcing attackers to perform a brute force attack on each separate piece of encrypted data.

To learn more about the mechanisms involved in encryption, read PKCS #5: Password-Based Cryptography Standard.

Figure 32 StandardPBEStringEncryptor, Part 2

Constructor Summary

`StandardPBEStringEncryptor()`
Creates a new instance of `StandardPBEStringEncryptor`.

Method Summary

`java.lang.String`	`decrypt(java.lang.String encryptedMessage)` Decrypts a message using the specified configuration.
`java.lang.String`	`encrypt(java.lang.String message)` Encrypts a message using the specified configuration.
`void`	`initialize()` Initialize the encryptor.
`boolean`	`isInitialized()` Returns true if the encryptor has already been initialized, false if not. Initialization happens:
`void`	`setAlgorithm(java.lang.String algorithm)` Sets the algorithm to be used for encryption, like `PBEWithMD5AndDES`.
`void`	`setConfig(PBEConfig config)` Sets a `PBEConfig` object for the encryptor.
`void`	`setKeyObtentionIterations(int keyObtentionIterations)` Set the number of hashing iterations applied to obtain the encryption key.
`void`	`setPassword(java.lang.String password)` Sets the password to be used.

NOTE The algorithms supported by the Security SDK API for PBE are PBEWithMD5AndDES, PBEWithMD5AndTripleDES, PBEWithSHA1AndDESede, and PBEWithSHA1AndRC2_40.

NOTE The Security SDK API also provides utility classes, such as `TextEncryptor` and `StrongTextEncryptor`, which are wrappers around the `StandardPBEStringEncryptor` class.

PBE Byte Encryption

The `StandardPBEByteEncryptor` class (see Figure 33 on page 50 and Figure 34 on page 50) provides interfaces to encrypt bytes using a password.

Figure 33 StandardPBEByteEncryptor, Part 1

com.comverse.rtbd.security.sdk.crypto.pbe

Class StandardPBEByteEncryptor

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.pbe.StandardPBEByteEncryptor
```

All Implemented Interfaces:
ByteEncryptor, PBEByteEncryptor

---

public final class **StandardPBEByteEncryptor**
extends java.lang.Object
implements PBEByteEncryptor

Standard implementation of the `PBEByteEncryptor` interface. This class lets the user specify the algorithm to be used for encryption, the password to use, and the number of hashing iterations that will be applied for obtaining the encryption key.

This class is *thread-safe*.

Figure 34 StandardPBEByteEncryptor, Part 2

Constructor Summary

`StandardPBEByteEncryptor()`	Creates a new instance of `StandardPBEByteEncryptor`.

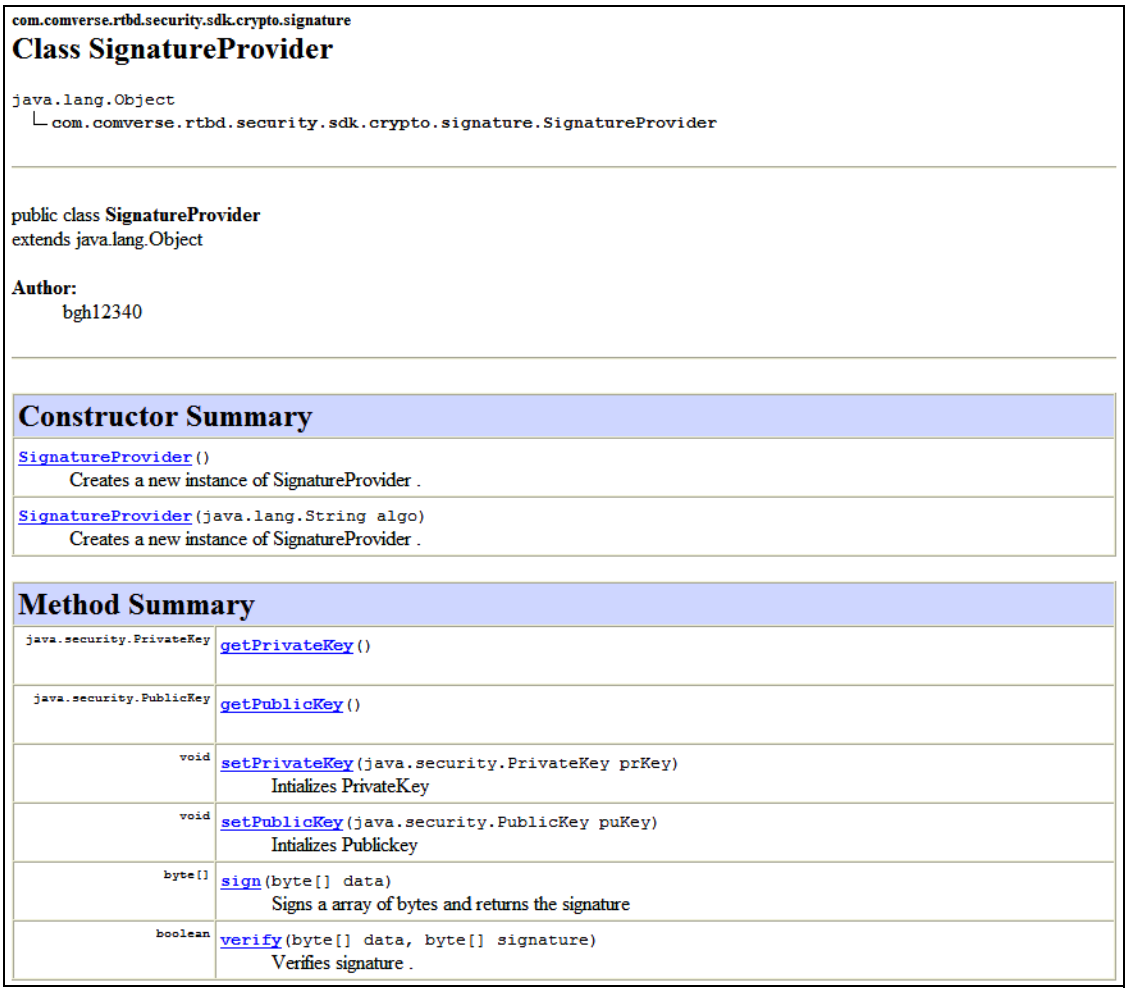
Method Summary

`byte[]`	`decrypt(byte[] encryptedMessage)` Decrypts a message using the specified configuration.
`byte[]`	`encrypt(byte[] message)` Encrypts a message using the specified configuration.
`void`	`initialize()` Initialize the encryptor.
`boolean`	`isInitialized()` Returns true if the encryptor has already been initialized, false if not. Initialization happens:
`void`	`setAlgorithm(java.lang.String algorithm)` Sets the algorithm to be used for encryption, like `PBEWithMD5AndDES`.
`void`	`setConfig(PBEConfig config)` Sets a `PBEConfig` object for the encryptor.
`void`	`setKeyObtentionIterations(int keyObtentionIterations)` Set the number of hashing iterations applied to obtain the encryption key.
`void`	`setPassword(java.lang.String password)` Sets the password to be used.

Digital Signatures

The `SignatureProvider` class (see Figure 35) provides interfaces for digitally signing data and verifying the signature associated with given data.

Figure 35 SignatureProvider



com.comverse.rtbd.security.sdk.crypto.signature

Class SignatureProvider

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.signature.SignatureProvider
```

public class **SignatureProvider**
extends java.lang.Object

Author:
bgh12340

Constructor Summary

`SignatureProvider()` Creates a new instance of SignatureProvider .
`SignatureProvider(java.lang.String algo)` Creates a new instance of SignatureProvider .

Method Summary

java.security.PrivateKey	`getPrivateKey()`
java.security.PublicKey	`getPublicKey()`
void	`setPrivateKey(java.security.PrivateKey prKey)` Intializes PrivateKey
void	`setPublicKey(java.security.PublicKey puKey)` Intializes Publickey
byte[]	`sign(byte[] data)` Signs a array of bytes and returns the signature
boolean	`verify(byte[] data, byte[] signature)` Verifies signature .

NOTE For digital signatures, the algorithms supported are RSAwithMD5 and SHA1WithRSA. The private key is used for signing the data and the public key is used for verifying data.

Message Digests

`StandardStringDigester` and `StandardByteDigester` classes provide interfaces for creating and verifying message digests (MAC). See the following figures:

- Figure 36, “StandardStringDigester, Part 1,” on page 52
- Figure 37, “StandardStringDigester, Part 2,” on page 53
- Figure 38, “StandardStringDigester, Part 3,” on page 54
- Figure 39, “StandardByteDigester, Part 1,” on page 54
- Figure 40, “StandardByteDigester, Part 2,” on page 55

Figure 36 StandardStringDigester, Part 1

com.comverse.rtbd.security.sdk.crypto.digest

Class StandardStringDigester

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.digest.StandardStringDigester
```

All Implemented Interfaces:
StringDigester

---

public final class **StandardStringDigester**
extends java.lang.Object
implements StringDigester

Standard implementation of the `StringDigester` interface. This class lets the user specify the algorithm to be used for creating digests, the size of the random salt to be applied, and the number of times the hash function will be applied (iterations).

This class avoids byte-conversion problems related to the fact of different platforms having different default charsets, and returns digests in the form of BASE64-encoded ASCII Strings.

This class is *thread-safe*.

Configuration

The algorithm, salt size and iterations can take values in any of these ways:

- Using its default values.
- Setting a `com.comverse.rtbd.security.sdk.crypto.digest.DigesterConfig` object which provides new configuration values.
- Calling the corresponding `setAlgorithm`, `setSaltSizeBytes` or `setIterations` methods.

And the actual values to be used for initialization will be established by applying the following priorities:

1. First, the default values are considered.
2. Then, if a `com.comverse.rtbd.security.sdk.crypto.digest.DigesterConfig` object has been set with `setConfig`, the non-null values returned by its `getX` methods override the default values.
3. Finally, if the corresponding `setX` method has been called on the digester itself for any of the configuration parameters, the values set by these calls override all of the above.

Initialization

Before it is ready to create digests, an object of this class has to be *initialized*. Initialization happens:

- When `initialize` is called.
- When `digest` or `matches` are called for the first time, if `initialize` has not been called before.

Once a digester has been initialized, trying to change its configuration (algorithm, salt size or iterations) will result in an `AlreadyInitializedException` being thrown.

Figure 37 StandardStringDigester, Part 2

Usage

A digester may be used in two different ways:

- For *creating digests*, by calling the `digest` method.
- For *matching digests*, this is, checking whether a digest corresponds adequately to a digest (as in password checking) or not, by calling the `matches` method.

The steps taken for creating digests are:

1. The String message is converted to a byte array.
2. A random salt of the specified size is generated (see `SaltGeneration`).
3. The salt bytes are added to the message.
4. The hash function is applied to the salt and message altogether, and then to the results of the function itself, as many times as specified (iterations).
5. The *undigested* salt and the final result of the hash function are concatenated.
6. The result of the concatenation is encoded in BASE64 and returned as an ASCII String.

Put schematically in bytes:

- DIGEST = `|S|..(ssb)..|S|X|X|X|...|X|`
  - `S`: salt bytes (plain, not digested).
  - `ssb`: salt size in bytes.
  - `X`: bytes resulting from hashing (see below).
- `|X|X|X|...|X|` = *H*`(`*H*`(`*H*`(..(it)..`*H*`(Z|Z|Z|...|Z|))))`
  - *H*: Hash function (algorithm).
  - `it`: Number of iterations.
  - `Z`: Input for hashing (see below).
- `|Z|Z|Z|...|Z|` = `|S|..(ssb)..|S|M|M|M...|M|`
  - `S`: salt bytes (plain, not digested).
  - `ssb`: salt size in bytes.
  - `M`: message bytes.

Two digests created for the same message will always be different (except in the case of random salt coincidence). Because of this, the result of the `digest` method contains both the *undigested* salt and the digest of the (salt + message), so that another digest operation can be performed with the same salt on a different message to check if both messages match (all of which will be managed automatically by the `matches` method).

To learn more about the mechanisms involved in digest creation, read PKCS #5: Password-Based Cryptography Standard.

Figure 38 StandardStringDigester, Part 3

Constructor Summary

StandardStringDigester()	Creates a new instance of StandardStringDigester.

Method Summary

java.lang.String	digest(java.lang.String message) Performs a digest operation on a String message.
void	initialize() Initialize the digester.
boolean	isInitialized() Returns true if the digester has already been initialized, false if not. Initialization happens:
boolean	matches(java.lang.String message, java.lang.String digest) Checks a message against a given digest.
void	setAlgorithm(java.lang.String algorithm) Sets the algorithm to be used for hashing, like MD5 or SHA-1.
void	setConfig(DigesterConfig config) Sets a com.comverse.rtbd.security.sdk.crypto.digest.DigesterConfig object for the digester.
void	setIterations(int iterations) Set the number of times the hash function will be applied recursively.
void	setSaltSizeBytes(int saltSizeBytes) Sets the size of the random salt to be used to compute the digest.

Figure 39 StandardByteDigester, Part 1

com.comverse.rtbd.security.sdk.crypto.digest

Class StandardByteDigester

java.lang.Object
  └ com.comverse.rtbd.security.sdk.crypto.digest.StandardByteDigester

All Implemented Interfaces:
ByteDigester

---

public final class **StandardByteDigester**
extends java.lang.Object
implements ByteDigester

Standard implementation of the ByteDigester interface. This class lets the user specify the algorithm to be used for creating digests, the size of the random salt to be applied, and the number of times the hash function will be applied (iterations).

This class is *thread-safe*.

Figure 40 StandardByteDigester, Part 2

Constructor Summary

	`StandardByteDigester()` Creates a new instance of `StandardByteDigester`.

Method Summary

`byte[]`	`digest(byte[] message)` Performs a digest operation on a byte array message.
`void`	`initialize()` Initialize the digester.
`boolean`	`isInitialized()` Returns true if the digester has already been initialized, false if not. Initialization happens:
`boolean`	`matches(byte[] message, byte[] digest)` Checks a message against a given digest.
`void`	`setAlgorithm(java.lang.String algorithm)` Sets the algorithm to be used for digesting, like `MD5` or `SHA-1`.
`void`	`setConfig(DigesterConfig config)` Sets a `com.comverse.rtbd.security.sdk.crypto.digest.DigesterConfig` object for the digester.
`void`	`setIterations(int iterations)` Set the number of times the hash function will be applied recursively.
`void`	`setSaltSizeBytes(int saltSizeBytes)` Sets the size of the random salt to be used to compute the digest.

Utility API

The Utility API provides a set of utility classes.

ConfigUtils

The `ConfigUtils` class (see Figure 41) provides wrapper interfaces for the most frequently used operations. The most important among them is the interface for retrieving the Security Server certificate. This certificate is required for `HTTPS` communication with the Security Server.

Figure 41 ConfigUtils

com.comverse.rtbd.security.sdk.commons.util

Class ConfigUtils

```
java.lang.Object
  └ com.comverse.rtbd.security.sdk.commons.util.ConfigUtils
```

public class **ConfigUtils**
extends java.lang.Object

Constructor Summary

ConfigUtils()

Method Summary

static java.lang.String	getJaasConfig()
static java.lang.String	getProperty(java.lang.String key)
static java.lang.String	getProperty(java.lang.String key, java.lang.String defalutValue)
static boolean	importSecurityCertificate()
static boolean	isPeerSSIPSet()
static boolean	isStubModeEnabled()
static void	setAlias(java.lang.String cetAlias)

Importing a Certificate from the Security Server/Unified Platform Manager

A Security Server web service needs to be invoked using the HTTPS protocol. The HTTPS protocol mandates that a certificate be imported from the server to establish communication with the server. The certificate used in the Security Server is a self-signed certificate and corresponding private key for SSL communication. The certificate and private key get generated when the Unified Platform Manager/Security Server is installed. The client application has to import the certificate from the machine. It can be retrieved using a call in the Security SDK library as follows:

```
ConfigUtils.importSecurityCertificate();
```

This is a standard Java API call, and it automates the retrieval and storing of the certificate in the keystore. If the API call is not used, then the application has to manually add the certificate to the keystore using the `keytool` command.

The Security Server keystore is in the following location:

```
/home/jboss/server/default/conf/upsec.keystore
```

The certificate in this keystore can be exported using the Java `keytool` utility, as follows:

```
$JAVA_HOME/bin/keytool -export -alias secserv -keystore
   /home/jboss/server/default/conf/upsec.keystore -rfc
   -file sec_certificate -storepass security
```

The `sec_certificate` file will contain the certificate, which client applications can import to the keystore configured by the system property `javax.net.ssl.trustStore`. The certificate can be exported either in PEM or DER format, described below. There will not be any certificate chains.

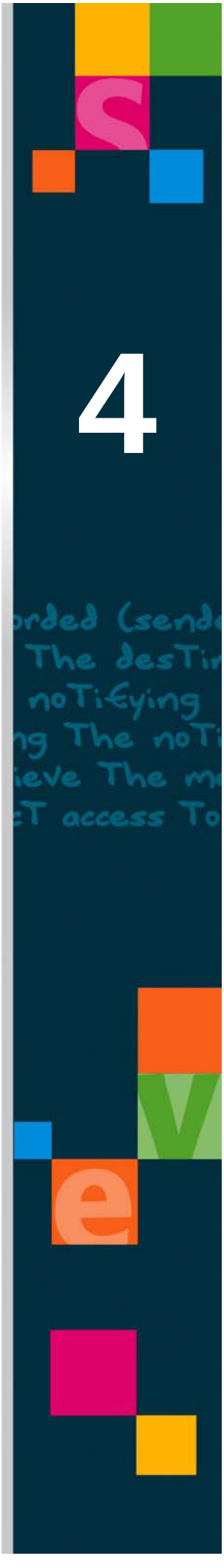
- **PEM Format**: Privacy Enhanced Mail (PEM) is a file format used to hold digital certificates. PEM format is a refinement of Base64 encoding. It is defined in RFC1421 for use in Privacy Enhanced Mail (PEM), hence its name.

  This format can contain all of the private keys (RSA and DSA), public keys (RSA and DSA) and (x509) certificates. It is the default format for OpenSSL. It stores data in Base64 encoded DER format, surrounded by ASCII headers, so it is suitable for text mode transfers between systems.
- **DER Format**: Distinguished Encoding Rules (DER) is a message transfer syntax specified by the ITU in X.690. DER provides for exactly one way to encode an ASN.1 value. DER is intended for situations when a unique encoding is needed, such as in cryptography, and it ensures that a data structure that needs to be digitally signed produces a unique serialized representation.

  This format can contain all of the private keys, public keys, and certificates. It is stored according to the ASN1 DER format. It is headerless. (PEM is text header wrapped DER.) It is the default format for most browsers.

Chapter 4

Security Server Web Services API



Overview

The Security Server Web Services API provides access to operations for the Security Server. Instead of executing these operations from the Security Server command line interface (CLI) or from the graphical user interface (GUI), the operations can be invoked using the Security Server Web Services API. A subset of operations (Identity Management operations) is exposed through the API.

The expectation is that you have an in-depth understanding of the Identity Management concepts and commands discussed in the *Security Platform Operations Guide*. Detailed information about security realms, groups, users, and roles provided in the *Security Platform Operations Guide* is not duplicated here.

Developers using this API need to know how to make a web service call. This can be done in many ways, and details are outside the scope of this document. Two helpful references are as follows:

`http://docs.jboss.org/jbossas/getting_started/v4/html/ws.html`

`http://e-docs.bea.com/wls/docs81/webserv/example.html`

Prerequisites for Using the Web Services API

The following are prerequisites for using the Security Server Web Services API:

1. Client applications must create a special user in the UPSEC security realm to perform provisioning using the Security Server Web Services API. This application-specific user is the administrative user for the application. This admin user and other users, along with roles and groups, are provisioned using batch provisioning. For information on batch provisioning, see the "Bulk Account Management Operations" section in the "Identity Management" chapter of the *Security Platform Operations Guide*.
2. To access all security operations via the Security Server Web Services API, a user/application must have the WEBADMIN role (that is, belong to a group that has the WEBADMIN role associated with it). This role is created as part of the default data in the Security Server database.
3. To limit a user to a specific operation or set of operations, specific roles can be associated with a user (that is, associated with a group to which the user belongs). These roles must be created and assigned to appropriate groups. They are not available as part of the default data in the Security Server database. Table 10 describes the roles required to authorize various types of access to methods in the API.

Table 10 Roles Required for Web Services API Methods

Method Name	Role Required
`addUser`	ADD_USER_ROLE
`removeUser`	REMOVE_USER_ROLE
`updateUser`	UPDATE_USER_ROLE
`lockUser`	UPDATE_USER_ROLE
`unlockUser`	UPDATE_USER_ROLE
`enableUser`	UPDATE_USER_ROLE

Table 10 Roles Required for Web Services API Methods (Continued)

Method Name	Role Required
disableUser	UPDATE_USER_ROLE
getUser	GET_USER_ROLE
addRole	ADD_ROLE_ROLE
removeRole	REMOVE_ROLE_ROLE
updateRole	UPDATE_ROLE_ROLE
getRole	GET_ROLE_ROLE
addRealm	ADD_REALM_ROLE
removeRealm	REMOVE_REALM_ROLE
updateRealm	UPDATE_REALM_ROLE
getRealm	GET_REALM_ROLE
addGroup	ADD_GROUP_ROLE
removeGroup	REMOVE_GROUP_ROLE
updateGroup	UPDATE_GROUP_ROLE
getGroup	GET_GROUP_ROLE
resetPassword	RESET_PASSWORD_ROLE
listUsers	LIST_USERS_ROLE
listRoles	LIST_ROLES_ROLE
listRealms	LIST_REALMS_ROLE
listGroups	LIST_GROUPS_ROLE

About the API Methods

The information below pertains to all the Security Server Web Services API methods discussed in "Security Server Web Services API Methods" on page 65.

Obtaining the Token

All the API methods require a token to be passed in. The token refers to the SAML authentication response returned after a successful authentication. The token can be obtained in one of two ways:

- Use the SDK API to log in and get the token.
- Make a web service call and get the token in response and then start using the Security Server Web Services API method calls.

Figure 42 on page 63 shows a code example. *This code example is for demonstration purposes only. It does not represent actual code to be used.*

Figure 42 Code Example for Obtaining the Token

```
package com.comverse.rtbd.sec.test.idm;

import java.io.IOException;
import java.io.StringReader;

import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.Namespace;
import org.jdom.input.SAXBuilder;

import com.comverse.rtbd.security.sdk.commons.constants.Constants;
import com.comverse.rtbd.security.sdk.commons.util.ConfigUtils;
import com.comverse.rtbd.security.sdk.commons.util.WebServiceUtil;
import com.comverse.rtbd.security.sdk.idm.saml.SAMLUtils;
import com.comverse.rtbd.security.sdk.idm.saml.SSOToken;


public class WebServiceTest {

  /**
   * @param args
   */
  public static void main(String[] args) {
    // System.setProperty("security.server.ip", "10.210.156.165");
    System.setProperty("security.server.ip", "208.48.158.245");

    System.setProperty("java.security.auth.login.config", "./conf/jaas.config");
    System.setProperty("javax.net.ssl.trustStore", "./conf/keystore");
    System.setProperty("javax.net.ssl.trustStore-pwd", "security");
    System.setProperty("security.client.context", "Client_Security");
    System.setProperty("security.webservice.stack", "weblogic");
    System.setProperty("security.sdk.mode", "false");
    System.setProperty("bea.home", "./conf");
    System.setProperty("weblogic.webservice.client.ssl.strictcertchecking",
      "false");
    System.setProperty("weblogic.webservice.client.ssl.adapterclass",
      "weblogic.webservice.client.JSSEAdapter");

    //String NS_URI = "https://org.comverse.rtbd.sec/webservice/auth";
    QName XSD_STRING = new QName("http://www.w3.org/2001/XMLSchema","string");

    try{

      String user = "trivuser";
      String realm= "TRIVNET";
      String password = "pa$$w0rd";


      /* import certificate */
      ConfigUtils.importSecurityCertificate();

      Call call = WebServiceUtil.getCallobject(Constants.IDM_NS_URI,
        Constants.IDM_WS_NAME, Constants.IDM_WS_PORT,
        "proxyLogin", Constants.IDM_ENDPOINT,Constants.SECURED_SCHEME);

      call.addParameter("String_1", XSD_STRING,ParameterMode.IN);
      call.addParameter("String_2", XSD_STRING,ParameterMode.IN);
      call.addParameter("String_3", XSD_STRING,ParameterMode.IN);
```

```
        call.setReturnType(XSD_STRING);

        Object ret = call.invoke(new Object[]{user,password,realm});

        System.out.println("Response from webservice->>>\n" +(String)ret );
        String token= parseResponse((String)ret);
        SSOToken tok = (SSOToken) SAMLUtils.getToken(token);
        System.out.println(" user from the token -->  " +tok.getAuthUser());
        System.out.println(" roles from the token -->  " +tok.getRoles().toString());
      } catch (Exception ex) {
        ex.printStackTrace();
      }

    }

    private static String parseResponse(String response) {
      String params=null;
      Document responseDoc=null;
      try {
        SAXBuilder builder= new SAXBuilder();
        StringReader sr =new StringReader(response);
        responseDoc = builder.build(sr);
        Element root_element=responseDoc.getRootElement();
        Namespace ns =
           Namespace.getNamespace("https://org.comverse.rtbd.sec/webservice/auth");

       if(root_element.getChild("Token",ns)!=null){
           params=root_element.getChild("Token",ns).getText();
          }
      } catch (JDOMException e1) {
        System.out.println("Error building the XML " + e1.getMessage());
      } catch (IOException e1) {
        System.out.println("Error building the XML " + e1.getMessage());
      }
       return params;
    }

}
```

Handling Exceptions

Two types of exceptions can be thrown from the server:

- The standard `java.rmi.RemoteException`.

  All remote methods in a web service throw this standard exception. This allows exceptions that arise from communications or runtime difficulties to propagate back to the caller or client.
- A checked, user-defined exception (`SSOException`).

Client applications need to handle the `SOAPFaultException` to get the message and detail object of the exception thrown from the server.

Security Server Web Services API Methods

The expectation is that you are thoroughly familiar with all Identity Management concepts and commands discussed in the *Security Platform Operations Guide*.

In this chapter, the XML input and output strings are formatted for readability.

addUser

The `addUser` method creates a user account in the given security realm. The required and optional data for creating a user is passed in the XML-formatted `userDetails` input string.

In the input string, you can provide data for custom attributes (if any) to be defined at the user level, designate which group(s) the user belongs to, indicate whether the user will be forced to change his/her password at the next login, assign the user's priority group (from among the groups the user belongs to), and set the account's state.

If you do not designate the group(s) for the user, by default the user will belong to the DEFAULT group for the security realm. If no priority group is assigned, by default the user's priority group will be the DEFAULT group for the security realm.

The account state indicates whether the user account will be purged after it has been inactive for a certain number of days. (ENABLED means the user account will be purged; DISABLED means it will not be purged.) Purged accounts are deleted from the `SEC_IDM_USER` database table and inserted in the `SEC_IDM_PURGED_USERS` table after the number of days set for the configurable `user.idletime` system property. This property is set in the `application.properties` file located in the `$JBOSS_HOME/conf/`directory.

Method Signature

```
public String addUser(String token, String userDetails)
    throws RemoteException, SSOException;
```

userDetails Input String Example

```
<?xml version="1.0" encoding="UTF-8"?>
<User name="testuser">
    <FirstName>firstname</FirstName>
    <MiddleName>middlename</MiddleName>
    <LastName>lastname</LastName>
    <Password>test@123</Password>
    <Realm>testrealm</Realm>
    <Phone>9999999999</Phone>
    <Extension>1234</Extension>
    <Department>department</Department>
    <Groups>
        <Group name="GROUP3"/>
```

```
        <Group name="GROUP1"/>
        <Group name="GROUP2"/>
    </Groups>
    <Attributes>
        <Attribute name="one" value="1"/>
        <Attribute name="two" value="2"/>
    </Attributes>
    <Email>xyz.abc@comverse.com</Email>
    <ForcePasswordChange>true</ForcePasswordChange>
    <PriorityGroup>group1</PriorityGroup>
    <AccountState>DISABLED</AccountState>
</User>
```

Required:

- User name
- Realm
- FirstName
- LastName
- Password (must comply with the password policy of the given realm)
- If Attributes (optional) are included in the input string, both the Attribute name and value are required.

Restrictions:

- The specified realm must already have been created.
- If Groups are included in the input string, the groups must already have been created.

addRealm

The `addRealm` method creates a security realm. The required and optional data for creating a realm is passed in the XML-formatted `realmDetails` input string.

In the input string, you can provide data to specify various parts of the security realm's password policy (if you do not want the default password policy to be used) and data for custom attributes (if any) to be defined at the realm level. All users provisioned in the security realm will inherit the realm's attributes. Also, passwords for all users provisioned in the security realm must comply with the realm's password policy.

Creating a security realm automatically creates a DEFAULT group for the realm.

Method Signature

```
public String addRealm(String token, String realmDetails)
    throws RemoteException, SSOException;
```

realmDetails Input String Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Realm name="testrealm">
    <ShortDisplay>realmfortest</ShortDisplay>
    <DisplayValue>testrealm</DisplayValue>
    <Attributes>
```

```
        <Attribute name="one" value="1"/>
        <Attribute name="two" value="2"/>
    </Attributes>
    <PasswordPolicy>
        <MinLength>6</MinLength>
        <MaxLength>20</MaxLength>
        <MinAlpha>1</MinAlpha>
        <MinAlphaLower>0</MinAlphaLower>
        <MinAlphaUpper>0</MinAlphaUpper>
        <MinOther>1</MinOther>
        <MinDiff>1</MinDiff>
        <MinAge>1</MinAge>
        <MaxAge>1</MaxAge>
        <MaxExpired>1</MaxExpired>
        <HistExpire>1</HistExpire>
        <HistSize>2</HistSize>
        <MaxRetries>4</MaxRetries>
        <LockInterval>40</LockInterval>
    </PasswordPolicy>
</Realm>
```

Required:

- Realm name
- If Attributes (optional) are included in the input string, both the Attribute name and value are required.

addRole

The `addRole` method creates a security role. The required and optional data for creating a role is passed in the XML-formatted `roleDetails` input string.

Method Signature

```
public String addRole(String token, String roleDetails)
    throws RemoteException, SSOException;
```

roleDetails Input String Example

```
<? xml version="1.0" encoding="UTF-8"?>
<Role name="testrole">
    <ShortDisplay>testrole</ShortDisplay>
    <DisplayValue>testrole</DisplayValue>
</Role>
```

`Required:`

- Role name

addGroup

The `addGroup` method creates a group in a given security realm. The required and optional data for creating a group is passed in the XML-formatted `groupDetails` input string.

In the input string, you can specify the role(s) to associate with the group, data for custom attributes (if any) to be defined at the group level, and the session policy that specifies hard timeout/soft timeout values (in minutes) for users that belong to the group. All users who belong to the group will inherit the group's attributes.

Method Signature

```
public String addGroup(String token, String groupDetails)
    throws RemoteException, SSOException;
```

groupDetails Input String Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Group name="testgroup">
    <Realm>testrealm</Realm>
    <ShortDisplay>testgroup</ShortDisplay>
    <DisplayValue>testgroup</DisplayValue>
    <Roles>
        <Role name="ADMIN"/>
        <Role name="GUEST"/>
    </Roles>
    <Attributes>
        <Attribute name="one" value="1"/>
        <Attribute name="two" value="2"/>
    </Attributes>
    <SessionPolicy>
        <HardTimer>1222200</HardTimer>
        <SoftTimer>111111</SoftTimer>
    </SessionPolicy>
</Group>
```

Required:

- Group name
- Realm
- If Attributes (optional) are included in the input string, both the Attribute name and value are required.

Restrictions:

- The specified realm must already have been created.

removeUser

The `removeUser` method deletes a user account from the given security realm. (The `secadmin` user account cannot be deleted. That is the account for the security administrator.)

Method Signature

```
public String removeUser(String token, String userName, String realmName)
    throws RemoteException, SSOException;
```

removeRealm

The `removeRealm` method deletes a security realm. (The UPSEC security realm cannot be deleted. That realm contains the administrator user and users who can perform actions on the Security Server and Unified Platform.)

Deleting a security realm deletes all user accounts and groups defined for the realm, along with the realm itself.

Method Signature

```
public String removeRealm(String token, String realmName)
    throws RemoteException, SSOException;
```

removeGroup

The `removeGroup` method deletes a group from the given security realm.

Method Signature

```
public String removeGroup(String token, String realmName, String groupName)
    throws RemoteException, SSOException;
```

removeRole

The `removeRole` method deletes a security role.

Method Signature

```
public String removeRole(String token, String roleName)
    throws RemoteException, SSOException;
```

updateUser

The `updateUser` method updates the details for a user account provisioned in the given security realm. The required and optional data for updating most details for a user account is passed in the XML-formatted `userDetails` input string. For example, to update a user's groups, include all the groups for the user in the input string. (The update operation replaces existing data and does not append to it.) In addition, using the `attribOp` input parameter, you can update any custom attributes defined at the user level.

A user password cannot be updated using the `updateUser` method. Instead, use the `resetPassword` method to reset a user's password. See "resetPassword" on page 73.

Method Signature

```
public String updateUser(String token, String userDetails, String attribOp)
    throws RemoteException, SSOException;
```

userDetails Input String Example

```
<?xml version="1.0" encoding="UTF-8"?>
<User name="testuser">
    <FirstName>firstname</FirstName>
    <MiddleName>middlename</MiddleName>
    <LastName>lastname</LastName>
    <Realm>testrealm</Realm>
    <Phone>9999999999</Phone>
    <Extension>1234</Extension>
    <Department>department</Department>
    <Groups>
        <Group name="GROUP3"/>
        <Group name="GROUP1"/>
        <Group name="GROUP2"/>
    </Groups>
    <Attributes>
        <Attribute name="one" value="1"/> [Example if attribOp = add.]
        <Attribute name="two" value="2"/> [Example if attribOp = add.]
    </Attributes>
    <Email>xyz.abc@comverse.com</Email>
    <ForcePasswordChange>true</ForcePasswordChange>
    <PriorityGroup>group1</PriorityGroup>
    <AccountState>ENABLED</AccountState>
</User>
```

Required:

- User name
- Realm

Restrictions:

- If Groups are included in the input string, the groups must already have been created.

Other Input Parameters for updateUser

- **attribOp**: Operations for the user's attributes. Valid values are `add` and `remove`.
  - `add`: Adds user-level attributes. The attribute names and values are provided in the elements nested under `<Attributes>` in the `userDetails` input string.
  - `remove`: Removes existing user-level attributes. The attributes to remove are provided in the elements nested under `<Attributes>` in the `userDetails` input string. Only the key (that is, the attribute name but not the attribute value) is required for attribute removal.

updateRealm

The `updateRealm` method updates the details for a given security realm. The required and optional data for updating most details for the realm is passed in the XML-formatted `realmDetails` input string. In addition, using the `passpolicyOp` and `attribOp` input parameters, you can update the realm's password policy and custom attributes defined at the realm level.

Method Signature

```
public String updateRealm(String token, String realmDetails, String passpolicyOp,
    String attribOp)
    throws RemoteException, SSOException;
```

realmDetails Input String Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Realm name="testrealm">
    <ShortDisplay>realmmodified</ShortDisplay>
    <DisplayValue>realmmodified</DisplayValue>
    <Attributes>
        <Attribute name="one"/> [Example if attribOp = remove.]
        <Attribute name="two"/> [Example if attribOp = remove.]
    </Attributes>
    <PasswordPolicy>
        <MinLength>6</MinLength>
        <MaxLength>20</MaxLength>
        <MinAlpha>2</MinAlpha>
        <MinAlphaLower>0</MinAlphaLower>
        <MinAlphaUpper>0</MinAlphaUpper>
        <MinOther>2</MinOther>
        <MinDiff>1</MinDiff>
        <MinAge>1</MinAge>
        <MaxAge>2</MaxAge>
        <MaxExpired>3</MaxExpired>
        <HistExpire>1</HistExpire>
        <HistSize>4</HistSize>
        <RestrictedPasswords>password,comverse</RestrictedPasswords>
    </PasswordPolicy>
</Realm>
```

Required:

- Realm name

Other Input Parameters for updateRealm

- **passpolicyOp**: Operations for the realm's password policy. Valid values are `none`, `remove`, and `modify`.
  - `none`: Does no operation on the password policy.
  - `remove`: Removes the entire existing password policy and replaces it with the default password policy.

  - `modify`: Changes the values for various parts of the realm's existing password policy. The new values are provided in the elements nested under `<PasswordPolicy>` in the `realmDetails` input string.

- **attribOp**: Operations for the realm's attributes. Valid values are `add` and `remove`.
  - `add`: Adds realm-level attributes. The attribute names and values are provided in the elements nested under `<Attributes>` in the `realmDetails` input string.
  - `remove`: Removes existing realm-level attributes. The attributes to remove are provided in the elements nested under `<Attributes>` in the `realmDetails` input string. Only the key (that is, only the attribute name but not the attribute value) is required for attribute removal.

updateGroup

The `updateGroup` method updates the details for a group in the given security realm. The required and optional data for updating most details for a group is passed in the XML-formatted `groupDetails` input string. For example, to update the roles associated with a group, include all roles for the group in the input string. (The update operation replaces existing data and does not append to it.) In addition, using the `attribOp` input parameter, you can update any custom attributes defined at the group level.

Method Signature

```
public String updateGroup(String token, String groupDetails, String attribOp)
    throws RemoteException, SSOException;
```

groupDetails Input String Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Group name="testgroup">
    <Realm>testrealm</Realm>
    <ShortDisplay>testgroup</ShortDisplay>
    <DisplayValue>testgroup</DisplayValue>
    <Roles>
        <Role name="ADMIN"/>
        <Role name="GUEST"/>
    </Roles>
    <Attributes>
        <Attribute name="one" value="1"/> [Example if attribOp = add.]
        <Attribute name="two" value="2"/> [Example if attribOp = add.]
    </Attributes>
    <SessionPolicy>
        <HardTimer>1222200</HardTimer>
        <SoftTimer>111111</SoftTimer>
    </SessionPolicy>
</Group>
```

Required:

- Group name
- Realm

Restrictions:

- If Roles are included in the input string, the roles must already have been created.

Other Input Parameters for updateGroup

- **attribOp**: Operations for the group's attributes. Valid values are `add` and `remove`.
  - `add`: Adds group-level attributes. The attribute names and values are provided in the elements nested under `<Attributes>` in the `groupDetails` input string.
  - `remove`: Removes existing group-level attributes. The attributes to remove are provided in the elements nested under `<Attributes>` in the `groupDetails` input string. Only the key (that is, only the attribute name but not the attribute value) is required for attribute removal.

updateRole

The `updateRole` method updates the details for a role. The required and optional data for updating a role is passed in the XML-formatted `roleDetails` input string.

Method Signature

```
public String updateRole(String token, String roleDetails)
    throws RemoteException, SSOException;
```

roleDetails Input String Example

```
<? xml version="1.0" encoding="UTF-8"?>
<Role name="testrole">
    <ShortDisplay>testrole</ShortDisplay>
    <DisplayValue>testrole</DisplayValue>
</Role>
```

Required:

- Role name

resetPassword

The `resetPassword` method resets the password for a given user and returns an XML-formatted string that includes the new system-generated password.

Method Signature

```
public String resetPassword(String token, String userName, String realmName)
    throws RemoteException, SSOException;
```

Output String Example for resetPassword

```
<Response>
    <Status>SUCCESS</Status>
    <Message>Reset password successful</Message>
    <Password>W7h%</Password>
</Response>
```

getRealm

The `getRealm` method returns an XML-formatted string that provides details about a given security realm. The details include all groups provisioned for the realm, custom attributes defined at the realm level (if any), the realm's password policy, and whether the password policy is the default policy.

Because custom attributes can be defined at the security realm, group, and user levels, the potential exists for conflicts among attributes. See Appendix A, "Attribute Conflict-Resolution Rules," for information on how attribute conflicts are resolved.

Method Signature

```
public String getRealm(String token, String realmName)
    throws RemoteException, SSOException;
```

Output String Example for getRealm

```
<Response>
    <Message>Retrieving Realm Success</Message>
    <Status>SUCCESS</Status>
    <Realm name="TEST">
        <DisplayValue>test</DisplayValue>
        <ShortDisplay>test</ShortDisplay>
        <CreateDate>2009-05-18 21:44:37.0</CreateDate>
        <ModifyDate>2009-05-18 21:44:37.0</ModifyDate>
        <Groups>
            <Group name="DEFAULT_GROUP_TEST"/>
            <Group name="GROUP3"/>
            <Group name="GROUP1"/>
            <Group name="GROUP2"/>
        </Groups>
        <Attributes>
            <Attribute name="abcd" value ="1234"/>
            <Attribute name="xyz" value ="1234"/>
            <Attribute name="tttt" value ="1234"/>
            <Attribute name="xxxx" value ="1234"/>
        </Attributes>
        <PasswordPolicy>
            <MinLength>6</MinLength>
            <MaxLength>20</MaxLength>
            <MinAlpha>1</MinAlpha>
            <MinAlphaLower>0</MinAlphaLower>
            <MinAlphaUpper>0</MinAlphaUpper>
            <MinOther>1</MinOther>
            <MinDiff>2</MinDiff>
            <MinAge>0</MinAge>
            <MaxAge>4</MaxAge>
            <MaxExpired>2</MaxExpired>
            <HistExpire>2</HistExpire>
            <HistSize>4</HistSize>
```

```
            <LockInterval>30</LockInterval>
            <MaxRetries>3</MaxRetries>
            <IsDefault>Y</IsDefault>
        </PasswordPolicy>
    </Realm>
</Response>
```

getUser

The `getUser` method returns an XML-formatted string that provides details about a user in a given security realm.

The output details include the user's first and last names, whether the user account is locked to prevent logins, whether the user will be forced to change his/her password at the next login, the groups that the user belongs to, the user's priority group, and any custom attributes for the user. The custom attributes include those defined at the user level, those inherited from the realm, and those inherited from the groups that the user belongs to.

Because custom attributes can be defined at the security realm, group, and user levels, the potential exists for conflicts among attributes. See Appendix A, "Attribute Conflict-Resolution Rules," for information on how attribute conflicts are resolved.

Method Signature

```
public String getUser(String token, String userName, String realmName)
    throws RemoteException, SSOException;
```

Output String Example for getUser

```
<Response>
    <Message>Retrieving User Success</Message>
    <Status>SUCCESS</Status>
    <User id="user1">
        <FirstName>test</FirstName>
        <LastName>test</LastName>
        <LockStatus>N</LockStatus>
        <ForcePasswordChange>N</ForcePasswordChange>
        <Realm>TEST</Realm>
        <CreatedDate>2009-05-19 19:09:50.0</CreatedDate>
        <LastLogin>2009-05-19 19:09:50.0</LastLogin>
        <PriorityGroup>DEFAULT_GROUP_TEST</PriorityGroup>
        <AccountState>ENABLED</AccountState>
        <Groups>
            <Group name="DEFAULT_GROUP_TEST"/>
            <Group name="GROUP3"/>
            <Group name="GROUP1"/>
            <Group name="GROUP2"/>
        </Groups>
        <Attributes>
            <Attribute name="abcd" value ="1234"/>
            <Attribute name="xyz" value ="1234"/>
            <Attribute name="tttt" value ="1234"/>
```

```
            <Attribute name="xxxx" value ="1234"/>
        </Attributes>
    </User>
</Response>
```

getRole

The `getRole` method returns an XML-formatted string that provides details about a role. The details include the group(s) that the role is associated with.

Method Signature

```
public String getRole(String token, String roleName)
    throws RemoteException, SSOException;
```

Output String Example for getRole

```
<Response>
    <Message>Retrieving Role Success</Message>
    <Status>SUCCESS</Status>
    <Role name="ROLE1">
        <DisplayValue>role1</DisplayValue>
        <ShortDisplay>role1</ShortDisplay>
        <CreateDate>2009-05-19 19:07:41.0</CreateDate>
        <ModifyDate>2009-05-19 19:07:41.0</ModifyDate>
        <Groups>
            <Group name="DEFAULT_GROUP_TEST"/>
            <Group name="GROUP3"/>
            <Group name="GROUP1"/>
            <Group name="GROUP2"/>
        </Groups>
    </Role>
</Response>
```

getGroup

The `getGroup` method returns an XML-formatted string that provides details about a group in a given security realm. The details include the session policy that provides soft timeout and hard timeout values (in minutes) for users in the group, whether the session policy is the default policy, the roles associated with the group, the users who belong to the group, and custom attributes (if any) defined for the group. Users who belong to the group inherit the group's attributes.

Because custom attributes can be defined at the security realm, group, and user levels, the potential exists for conflicts among attributes. See Appendix A, "Attribute Conflict-Resolution Rules," for information on how attribute conflicts are resolved.

Method Signature

```
public String getGroup(String token, String realmName, String groupName
    throws RemoteException, SSOException;
```

Output String Example for getGroup

```
<Response>
    <Message>Retrieving Group Success</Message>
    <Status>SUCCESS</Status>
    <Group name="GROUP1">
        <DisplayValue>group1</DisplayValue>
        <ShortDisplay>group1</ShortDisplay>
        <Realm>TEST</Realm>
        <CreateDate>2009-05-19 19:07:22.0</CreateDate>
        <ModifyDate>2009-05-19 19:07:22.0</ModifyDate>
        <SessionPolicy>
            <SoftTimer>30</SoftTimer>
            <HardTimer>480</HardTimer>
            <IsDefault>Y</IsDefault>
        </SessionPolicy>
        <Roles>
            <Role name= "ADMIN"/>
            <Role name= "GUEST"/>
        </Roles>
        <Users>
            <User name="user2"/>
            <User name="user1"/>
        </Users>
        <Attributes>
            <Attribute name="abcd" value ="1234"/>
            <Attribute name="xyz" value ="1234"/>
            <Attribute name="tttt" value ="1234"/>
            <Attribute name="xxxx" value ="1234"/>
        </Attributes>
    </Group>
</Response>
```

listRealms

The `listRealms` method returns an XML-formatted string that provides details about all the security realms. The details include the groups provisioned for the realm, custom attributes defined for the realm (if any), the realm's password policy and whether the policy is the default password policy.

Method Signature

```
public String listRealms(String token)
    throws RemoteException, SSOException;
```

Output String Example for listRealms

```
<Response>
    <Message>Retrieving List of Realms Success</Message>
    <Status>SUCCESS</Status>
    <Realms>
```

```
<Realm name="UPSEC">
    <DisplayValue>UPSEC</DisplayValue>
    <ShortDisplay>SRl</ShortDisplay>
    <LanguageCode>1</LanguageCode>
    <ModifiedUser>installation</ModifiedUser>
    <CreateDate>2009-05-14 23:58:27.0</CreateDate>
    <ModifyDate>2009-05-14 23:58:27.0</ModifyDate>
    <Groups>
        <Group name="DEFAULT_GROUP_TEST"/>
        <Group name="GROUP3”/>
        <Group name="GROUP1"/>
        <Group name="GROUP2"/>
    </Groups>
    <Attributes>
        <Attribute name="abcd" value ="1234"/>
        <Attribute name="xyz" value ="1234"/>
        <Attribute name="tttt" value ="1234"/>
        <Attribute name="xxxx" value ="1234"/>
    </Attributes>
    <PasswordPolicy>
        <MinLength>6</MinLength>
        <MaxLength>20</MaxLength>
        <MinAlpha>1</MinAlpha>
        <MinAlphaLower>0</MinAlphaLower>
        <MinAlphaUpper>0</MinAlphaUpper>
        <MinOther>1</MinOther>
        <MinDiff>2</MinDiff>
        <MinAge>0</MinAge>
        <MaxAge>999</MaxAge>
        <MaxExpired>12</MaxExpired>
        <HistExpire>1</HistExpire>
        <HistSize>2</HistSize>
        <LockInterval>30</LockInterval>
        <MaxRetries>3</MaxRetries>
        <IsDefault>N</IsDefault>
    </PasswordPolicy>
</Realm>
<Realm name="TEST">
    <DisplayValue>test</DisplayValue>
    <ShortDisplay>test</ShortDisplay>
    <CreateDate>2009-05-18 21:44:37.0</CreateDate>
    <ModifyDate>2009-05-18 21:44:37.0</ModifyDate>
    <Groups>
        <Group name="DEFAULT_GROUP_TEST"/>
        <Group name="GROUP3"/>
        <Group name="GROUP1"/>
        <Group name="GROUP2"/>
```

```
        </Groups>
        <Attributes>
            <Attribute name="abcd" value ="1234"/>
            <Attribute name="xyz" value ="1234"/>
            <Attribute name="tttt" value ="1234"/>
            <Attribute name="xxxx" value ="1234"/>
        </Attributes>
        <PasswordPolicy>
            <MinLength>6</MinLength>
            <MaxLength>20</MaxLength>
            <MinAlpha>1</MinAlpha>
            <MinAlphaLower>0</MinAlphaLower>
            <MinAlphaUpper>0</MinAlphaUpper>
            <MinOther>1</MinOther>
            <MinDiff>2</MinDiff>
            <MinAge>0</MinAge>
            <MaxAge>4</MaxAge>
            <MaxExpired>2</MaxExpired>
            <HistExpire>2</HistExpire>
            <HistSize>4</HistSize>
            <LockInterval>30</LockInterval>
            <MaxRetries>3</MaxRetries>
            <IsDefault>Y</IsDefault>
        </PasswordPolicy>
    </Realm>
    <Realm name="TESTREALM">
        <DisplayValue>realmmodified</DisplayValue>
        <ShortDisplay>realmmodified</ShortDisplay>
        <CreateDate>2009-05-16 17:33:21.0</CreateDate>
        <ModifyDate>2009-05-16 17:58:22.0</ModifyDate>
        <Groups>
            <Group name="DEFAULT_GROUP_TEST"/>
            <Group name="GROUP3"/>
            <Group name="GROUP1"/>
            <Group name="GROUP2"/>
        </Groups>
        <Attributes>
            <Attribute name="abcd" value ="1234"/>
            <Attribute name="xyz" value ="1234"/>
            <Attribute name="tttt" value ="1234"/>
            <Attribute name="xxxx" value ="1234"/>
        </Attributes>
        <PasswordPolicy>
            <MinLength>6</MinLength>
            <MaxLength>20</MaxLength>
            <MinAlpha>2</MinAlpha>
            <MinAlphaLower>0</MinAlphaLower>
```

```
            <MinAlphaUpper>0</MinAlphaUpper>
            <MinOther>2</MinOther>
            <MinDiff>1</MinDiff>
            <MinAge>1</MinAge>
            <MaxAge>2</MaxAge>
            <MaxExpired>3</MaxExpired>
            <HistExpire>1</HistExpire>
            <HistSize>4</HistSize>
            <LockInterval>40</LockInterval>
            <MaxRetries>4</MaxRetries>
            <RestrictedPasswords>password,comverse</RestrictedPasswords>
            <IsDefault>N</IsDefault>
        </PasswordPolicy>
    </Realm>
  </Realms>
</Response>
```

listUsers

The `listUsers` method returns an XML-formatted string that provides details about the users provisioned in a given security realm. The details include the user name (user ID), user's first and last names, whether the user account is locked to prevent logins, whether the user will be forced to change his/her password at the next login, the group(s) that the user belongs to, the user's priority group, and custom attributes (if any) for the user. The custom attributes include those defined at the user level, those inherited from the realm, and those inherited from the group(s) that the user belongs to.

Method Signature

```
public String listUsers(String token, String realm)
    throws RemoteException, SSOException;
```

Output String Example for listUsers

```
<Response>
   <Message>Retrieving List of Users Success</Message>
   <Status>SUCCESS</Status>
   <Users>
      <User name="testuser">
         <FirstName>test</FirstName>
         <LastName>test</LastName>
         <LockStatus>N</LockStatus>
         <ForcePasswordChange>N</ForcePasswordChange>
         <Realm>TEST</Realm>
         <CreatedDate>2009-05-18 19:30:29.0</CreatedDate>
         <LastLogin>2009-05-18 20:40:59.0</LastLogin>
         <PriorityGroup>DEFAULT_GROUP_TEST</PriorityGroup>
         <Groups>
            <Group name ="DEFAULT_GROUP_TEST"/>
         </Groups>
```

```
            <Attributes>
            </Attributes>
        </User>
        <User name="user1">
            <FirstName>test</FirstName>
            <LastName>test</LastName>
            <LockStatus>N</LockStatus>
            <ForcePasswordChange>N</ForcePasswordChange>
            <Realm>TEST</Realm>
            <CreatedDate>2009-05-19 19:09:50.0</CreatedDate>
            <LastLogin>2009-05-19 19:09:50.0</LastLogin>
            <PriorityGroup>DEFAULT_GROUP_TEST</PriorityGroup>
            <Groups>
                <Group name="DEFAULT_GROUP_TEST"/>
                <Group name="GROUP3”/>
                <Group name="GROUP1"/>
                <Group name="GROUP2"/>
            </Groups>
            <Attributes>
                <Attribute name="abcd" value ="1234"/>
                <Attribute name="xyz" value ="1234"/>
                <Attribute name="tttt" value ="1234"/>
                <Attribute name="xxxx" value ="1234"/>
            </Attributes>
        </User>
        <User name="user2">
            <FirstName>test</FirstName>
            <LastName>test</LastName>
            <LockStatus>N</LockStatus>
            <ForcePasswordChange>N</ForcePasswordChange>
            <Realm>TEST</Realm>
            <CreatedDate>2009-05-19 19:10:12.0</CreatedDate>
            <LastLogin>2009-05-19 19:10:12.0</LastLogin>
            <PriorityGroup>GROUP1</PriorityGroup>
            <Groups>
                <Group name= "GROUP1"/>
            </Groups>
            <Attributes>
                <Attribute name="abcd" value ="1234"/>
                <Attribute name="xyz" value ="1234"/>
                <Attribute name="tttt" value ="1234"/>
                <Attribute name="xxxx" value ="1234"/>
            </Attributes>
        </User>
    </Users>
</Response>
```

listRoles

The `listRoles` method returns an XML-formatted string that provides details about all the security roles. The details include the group(s) with which the role is associated.

Method Signature

```
public String listRoles(String token)
    throws RemoteException, SSOException;
```

Output String Example for listRoles

```
<Response>
    <Message>Retrieving List of Roles Success</Message>
    <Status>SUCCESS</Status>
    <Roles>
        <Role name="ROLE2">
            <DisplayValue>role2</DisplayValue>
            <ShortDisplay>role2</ShortDisplay>
            <CreateDate>2009-05-19 19:07:43.0</CreateDate>
            <ModifyDate>2009-05-19 19:07:43.0</ModifyDate>
            <Groups>
                <Group name="GROUP3"/>
            </Groups>
        </Role>
        <Role name="ROLE1">
            <DisplayValue>role1</DisplayValue>
            <ShortDisplay>role1</ShortDisplay>
            <CreateDate>2009-05-19 19:07:41.0</CreateDate>
            <ModifyDate>2009-05-19 19:07:41.0</ModifyDate>
            <Groups>
                <Group name="DEFAULT_GROUP_TEST"/>
                <Group name="GROUP3”/>
                <Group name="GROUP1"/>
                <Group name="GROUP2"/>
            </Groups>
        </Role>
        <Role name="ADMIN">
            <DisplayValue>admin</DisplayValue>
            <ShortDisplay>admin</ShortDisplay>
            <LanguageCode>1</LanguageCode>
            <ModifiedUser>installation</ModifiedUser>
            <CreateDate>2009-05-14 23:58:27.0</CreateDate>
            <ModifyDate>2009-05-14 23:58:27.0</ModifyDate>
            <Groups>
                <Group name="DEFAULT_GROUP_TEST"/>
                <Group name="GROUP3”/>
                <Group name="GROUP1"/>
                <Group name="GROUP2"/>
            </Groups>
```

```
</Role>
<Role name="WEBADMIN">
    <DisplayValue>webadmin</DisplayValue>
    <ShortDisplay>webadmin</ShortDisplay>
    <LanguageCode>1</LanguageCode>
    <ModifiedUser>installation</ModifiedUser>
    <CreateDate>2009-05-14 23:58:27.0</CreateDate>
    <ModifyDate>2009-05-14 23:58:27.0</ModifyDate>
    <Groups>
        <name>DEFAULT_GROUP</name>
    </Groups>
</Role>
<Role name="ROLE3">
    <DisplayValue>role3</DisplayValue>
    <ShortDisplay>role3</ShortDisplay>
    <CreateDate>2009-05-19 19:07:45.0</CreateDate>
    <ModifyDate>2009-05-19 19:07:45.0</ModifyDate>
    <Groups>
        <Group name="DEFAULT_GROUP_TEST"/>
        <Group name="GROUP3”/>
        <Group name="GROUP1"/>
        <Group name="GROUP2"/>
    </Groups>
</Role>
<Role name="GUEST">
    <DisplayValue>guest</DisplayValue>
    <ShortDisplay>guest</ShortDisplay>
    <LanguageCode>1</LanguageCode>
    <ModifiedUser>installation</ModifiedUser>
    <CreateDate>2009-05-14 23:58:27.0</CreateDate>
    <ModifyDate>2009-05-14 23:58:27.0</ModifyDate>
    <Groups>
        <Group name="DEFAULT_GROUP_TEST"/>
        <Group name="GROUP3”/>
        <Group name="GROUP1"/>
        <Group name="GROUP2"/>
    </Groups>
</Role>
<Role name="TESTROLE">
    <DisplayValue>testrole</DisplayValue>
    <ShortDisplay>testrole</ShortDisplay>
    <CreateDate>2009-05-16 17:35:54.0</CreateDate>
    <ModifyDate>2009-05-16 17:35:54.0</ModifyDate>
    <Groups>
        <Group name="DEFAULT_GROUP_TEST"/>
        <Group name="GROUP3”/>
        <Group name="GROUP1"/>
```

```
            <Group name="GROUP2"/>
         </Groups>
      </Role>
      <Role name="BILLING_ROLE">
         <DisplayValue>billing_role</DisplayValue>
         <ShortDisplay>billing_role</ShortDisplay>
         <LanguageCode>1</LanguageCode>
         <ModifiedUser>installation</ModifiedUser>
         <CreateDate>2009-05-14 23:58:27.0</CreateDate>
         <ModifyDate>2009-05-14 23:58:27.0</ModifyDate>
         <Groups>
            <Group name="GROUP2"/>
         </Groups>
      </Role>
   </Roles>
</Response>
```

listGroups

The `listGroups` method returns an XML-formatted string that provides details about the groups in a given security realm. The details include the session policy that specifies hard timeout/soft timeout values (in minutes) for users that belong to the group and indicates whether the policy is the default policy, the role(s) associated with the group, the users who belong to the group, and custom attributes (if any) defined for the group.

Method Signature

```
public String listGroups(String token, String realmName)
   throws RemoteException, SSOException;
```

Output String Example for listGroups

```
<Response>
   <Message>Retrieving List of Groups Success</Message>
   <Status>SUCCESS</Status>
   <Groups>
      <Group name="GROUP2">
         <DisplayValue>group2</DisplayValue>
         <ShortDisplay>group2</ShortDisplay>
         <Realm>TEST</Realm>
         <CreateDate>2009-05-19 19:07:28.0</CreateDate>
         <ModifyDate>2009-05-19 19:07:28.0</ModifyDate>
         <SessionPolicy>
            <SoftTimer>30</SoftTimer>
            <HardTimer>480</HardTimer>
            <IsDefault>Y</IsDefault>
         </SessionPolicy>
         <Roles>
            <Role name="ADMIN"/>
            <Role name="GUEST"/>
```

```
        </Roles>
        <Users>
            <User name="user2"/>
            <User name="user1"/>
        </Users>
        <Attributes>
        </Attributes>
    </Group>
    <Group name="GROUP3">
        <DisplayValue>group3</DisplayValue>
        <ShortDisplay>group3</ShortDisplay>
        <Realm>TEST</Realm>
        <CreateDate>2009-05-19 19:08:11.0</CreateDate>
        <ModifyDate>2009-05-19 19:08:11.0</ModifyDate>
        <SessionPolicy>
            <SoftTimer>30</SoftTimer>
            <HardTimer>480</HardTimer>
            <IsDefault>Y</IsDefault>
        </SessionPolicy>
        <Roles>
            <Role name="ADMIN"/>
            <Role name="GUEST"/>
        </Roles>
        <Attributes>
        </Attributes>
    </Group>
    <Group name="GROUP1">
        <DisplayValue>group1</DisplayValue>
        <ShortDisplay>group1</ShortDisplay>
        <Realm>TEST</Realm>
        <CreateDate>2009-05-19 19:07:22.0</CreateDate>
        <ModifyDate>2009-05-19 19:07:22.0</ModifyDate>
        <SessionPolicy>
            <SoftTimer>30</SoftTimer>
            <HardTimer>480</HardTimer>
            <IsDefault>Y</IsDefault>
        </SessionPolicy>
        <Roles>
            <Role name="ADMIN"/>
            <Role name="GUEST"/>
        </Roles>
        <Users>
            <User name="user2"/>
            <User name="user1"/>
        </Users>
        <Attributes>
        </Attributes>
```

```
        </Group>
        <Group name="DEFAULT_GROUP_TEST">
            <DisplayValue>default group</DisplayValue>
            <Realm>TEST</Realm>
            <CreateDate>2009-05-18 21:44:37.0</CreateDate>
            <ModifyDate>2009-05-18 21:44:37.0</ModifyDate>
            <SessionPolicy>
                <SoftTimer>30</SoftTimer>
                <HardTimer>480</HardTimer>
                <IsDefault>Y</IsDefault>
            </SessionPolicy>
            <Roles>
                <Role name="ADMIN"/>
                <Role name="GUEST"/>
            </Roles>
            <Users>
                <User name="user2"/>
                <User name="user1"/>
            </Users>
            <Attributes>
            </Attributes>
        </Group>
    </Groups>
</Response>
```

lockUser

The `lockUser` method locks a user account, preventing logins.

Method Signature

```
public String lockUser(String token, String userName, String realmName)
    throws RemoteException, SSOException;
```

unlockUser

The `unlockUser` method unlocks a user account, enabling logins.

Method Signature

```
public String unlockUser(String token, String userName, String realmName)
    throws RemoteException, SSOException;
```

enableUser

The `enableUser` method sets the state of a given user's account to ENABLED. This state means that the account will be purged after a certain number of days of inactivity (the default is 30 days). The number of days of inactivity is specified by the configurable `user.idletime` system property. This property is set in the `application.properties` file located in the `$JBOSS_HOME/conf/` directory. Purged user accounts are deleted from the `SEC_IDM_USER` database table and inserted in the `SEC_IDM_PURGED_USERS` table.

Method Signature

```
public String enableUser(String token, String userName, String realmName)
    throws RemoteException, SSOException;
```

disableUser

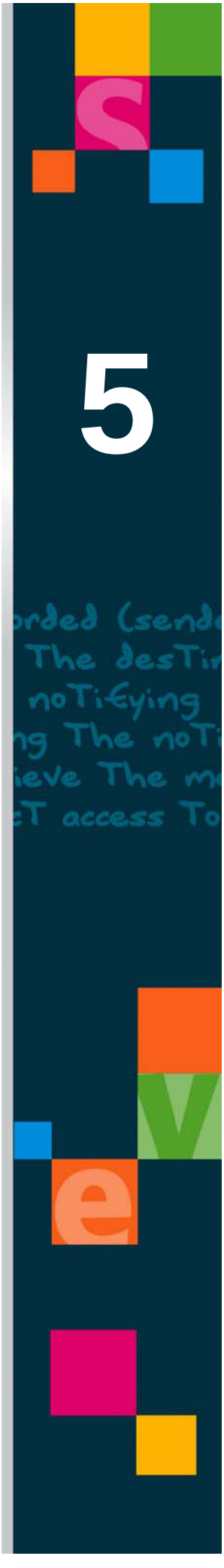
The `disableUser` method sets the state of a given user's account to DISABLED. This state means that the user account will not be purged because of inactivity.

Method Signature

```
public String disableUser(String token, String userName, String realmName)
    throws RemoteException, SSOException;
```

Chapter 5

Security SDK API Code Examples



Identity Management API Code Examples

This section provides code examples pertaining to identity management and demonstrates how client applications can perform various authentication operations. *These code examples are for demonstration purposes only. They do not represent actual code to be used.*

Login and Logout

The code example in Figure 43 demonstrates how client applications can use the `LoginContext` object to perform login and logout operations. In the example, the circled area indicates properties that must be set if the client application uses the WebLogic web services stack.

Figure 43 Login/Logout Code Example

```
package com.comverse.rtbd.security.sdk.test.IDM;

import javax.security.auth.login.LoginException;

import com.comverse.rtbd.security.sdk.commons.constants.ErrorCode;
import com.comverse.rtbd.security.sdk.idm.LoginContext;

public class LoginLogout {

    LoginLogout() {
    }

    public static void main(String args[]) {

        LoginContext ctx = null;

        System.setProperty("javax.net.ssl.trustStore", "./conf/keystore");
        System.setProperty("javax.net.ssl.trustStore-pwd", "security");
        System.setProperty("security.client.context", "Client_Security");
        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.73.17.114");

        System.setProperty("security.webservice.stack", "weblogic");
        System.setProperty("weblogic.webservice.client.ssl.strictcertchecking",
        "false");
        System.setProperty("weblogic.webservice.client.ssl.adapterclass",
        "weblogic.webservice.client.JSSEAdapter");

        try {

            //login operation
            ctx = new LoginContext("user7", "realm6", "password4");
            ctx.login();

            //logout operation
            ctx.logout();

        } catch (LoginException e) {

            System.out.println("Exception-->" + e.getMessage());
            System.out.println("##### ERROR CODE #####");
            ErrorCode code = ctx.getErrorCode();

            System.out.println("RESOURCE_NAME="+code.getName());
            System.out.println("MESSAGE="+code.getLocalizedMessage());



        }

    }
}
```

Client applications should set the `java.security.auth.login.config` virtual machine argument, which specifies the location of the JAAS configuration file.

Timer Management

The code example in Figure 44 demonstrates how client applications can obtain session timers associated with a particular user (user7 in this example) from the `LoginContext`.

Figure 44 Session Timers Code Example

```
package com.comverse.rtbd.security.sdk.test.IDM;

import javax.security.auth.login.LoginException;

public class SessionTimers {

    SessionTimers() {

    }

    public static void main(String args[]) {

        LoginContext ctx = null;

        System.setProperty("javax.net.ssl.trustStore", "./conf/keystore");
        System.setProperty("javax.net.ssl.trustStore-pwd", "security");
        System.setProperty("security.client.context", "Client_Security");
        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.73.17.114");

        try {
            ctx = new LoginContext("user7", "realm6", "password4");
            ctx.login();

            //obtain token object
            SSOToken tok = ctx.getToken();

            //extract sessionpolicy from  token object
            SessionPolicy pol = tok.getSessionTimeouts();

            System.out.println(" Hard Timer is " + pol.getHardTimer());
            System.out.println(" Soft Timer is " + pol.getSoftTimer());

        } catch (LoginException e) {
            e.printStackTrace();
        } catch (SSOException e) {
            ErrorCode code = ctx.getErrorCode();
            System.out.println("RESOURCE_NAME="+code.getName());
            System.out.println("MESSAGE="+code.getLocalizedMessage());
        }

    }

}
```

Attribute Management

The code example in Figure 45 demonstrates how client applications can add, modify, and delete attributes associated with a user (user7 in this example).

Figure 45 User Attributes Code Example

```
package com.comverse.rtbd.security.sdk.test.IDM;

import java.util.HashMap;

import javax.security.auth.login.LoginException;

import com.comverse.rtbd.security.sdk.commons.constants.ErrorCode;
import com.comverse.rtbd.security.sdk.exceptions.SSOException;
import com.comverse.rtbd.security.sdk.idm.LoginContext;

public class UserAttributes {

    UserAttributes() {

    }

    public static void main(String args[]) {

        LoginContext ctx = null;

        System.setProperty("javax.net.ssl.trustStore", "./conf/keystore");
        System.setProperty("javax.net.ssl.trustStore-pwd", "security");
        System.setProperty("security.client.context", "Client_Security");
        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.73.17.114");


        try {


            ctx = new LoginContext("user7", "realm6", "password4");
            ctx.login();

            //Adding User Attributes
            HashMap addMap = new HashMap();
            addMap.put("one", "1");
            addMap.put("two", "2");
            ctx.addUserAttributes(ctx.getSSOToken(), addMap);

            //Removing User Attributes
            HashMap removeMap = new HashMap();
            removeMap.put("two", "2");
            ctx.removeUserAttributes(ctx.getSSOToken(), removeMap);

            //Changing User Attributes
            HashMap changeMap = new HashMap();
            changeMap.put("one", "2");
            ctx.changeUserAttributes(ctx.getSSOToken(), changeMap);

        } catch (LoginException e) {
            e.printStackTrace();
        } catch (SSOException e) {

            ErrorCode code = ctx.getErrorCode();
            System.out.println("RESOURCE_NAME="+code.getName());
            System.out.println("MESSAGE="+code.getLocalizedMessage());
        }

    }

}
```

Token Management

The code example in Figure 46 demonstrates how client applications can (1) get the SSO token (string) from `LoginContext` and (2) reconstruct the `SSOToken` object from the token string using the `SAMLUtils` utility class.

Figure 46 SSO Token Code Example

```
package com.comverse.rtbd.security.sdk.test.IDM;

import javax.security.auth.login.LoginException;

import com.comverse.rtbd.security.sdk.exceptions.SSOException;
import com.comverse.rtbd.security.sdk.idm.LoginContext;
import com.comverse.rtbd.security.sdk.idm.saml.SAMLUtils;
import com.comverse.rtbd.security.sdk.idm.saml.SSOToken;

public class TokenManagement{

    TokenManagement(){
    }

    public static void main(String args[]) {

        LoginContext ctx = null;

        System.setProperty("javax.net.ssl.trustStore", "./conf/keystore");
        System.setProperty("javax.net.ssl.trustStore-pwd", "security");
        System.setProperty("security.client.context", "Client_Security");
        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.73.17.114");

         String sToken;
        try {

            ctx = new LoginContext("user7", "realm6", "password4");
            ctx.login();

            sToken = ctx.getSSOToken();

            //obtain an insatnce of SSOToken object
            SSOToken token = (SSOToken)SAMLUtils.getToken(sToken);

            System.out.println(" SessionID-->"+ token.getSessionId());
            System.out.println(" Token expiry time-->"+ token.getTokenExpiryTime());

        } catch (SSOException e) {
            e.printStackTrace();
        } catch (LoginException e) {
            e.printStackTrace();
        }

    }
}
```

Change Password

The code example in Figure 47 demonstrates how client applications can change the password of a particular user using the `LoginContext` object. Both old and new passwords must be supplied. The new password must comply with the password policy of the security realm in which the user is provisioned.

Figure 47 Change Password Code Example

```
package com.comverse.rtbd.security.sdk.test.IDM;

import javax.security.auth.login.LoginException;

import com.comverse.rtbd.security.sdk.commons.constants.ErrorCode;
import com.comverse.rtbd.security.sdk.exceptions.SSOException;
import com.comverse.rtbd.security.sdk.idm.LoginContext;

public class ChangePassword {

    ChangePassword() {
    }

    public static void main(String args[]) {

        LoginContext ctx = null;

        System.setProperty("javax.net.ssl.trustStore", "./conf/keystore");
        System.setProperty("javax.net.ssl.trustStore-pwd", "security");
        System.setProperty("security.client.context", "Client_Security");
        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.73.17.114");


        try {

            ctx = new LoginContext("user7", "realm6", "password4");
            ctx.login();

            //Changing password using SSOToken Object
            ctx.changePassword(ctx.getToken(), "password4", "password56");

            //Changing password using SSOToken in string format
            ctx.changePassword(ctx.getSSOToken(), "password56", "password2323");

        } catch (LoginException e) {
            ErrorCode code = ctx.getErrorCode();

            System.out.println("RESOURCE_NAME="+code.getName());
            System.out.println("MESSAGE="+code.getLocalizedMessage());
        } catch (SSOException e) {
            ErrorCode code = ctx.getErrorCode();

            System.out.println("RESOURCE_NAME="+code.getName());
            System.out.println("MESSAGE="+code.getLocalizedMessage());
        }

    }

}
```

Policy Management API Code Example

This section provides a code example pertaining to authorization policy management. The code example in Figure 48 and Figure 49 on page 97 demonstrates how an application can create an embedded PDP and use the application's authorization policy to determine whether access to a resource is authorized. *This code example is for demonstration purposes only. It does not represent actual code to be used.*

Figure 48 Policy Management Code Example, Part 1

```
package TestAuthorization;

import java.util.List;
import java.util.Set;
import java.util.TreeSet;


import javax.security.auth.login.LoginException;

import com.comverse.rtbd.security.sdk.commons.constants.Constants;
import com.comverse.rtbd.security.sdk.commons.util.ConfigUtils;
import com.comverse.rtbd.security.sdk.idm.LoginContext;
import com.comverse.rtbd.security.sdk.idm.saml.SAMLUtils;
import com.comverse.rtbd.security.sdk.idm.saml.SSOToken;
import com.comverse.rtbd.security.sdk.policy.client.EmbeddedPDPClient;
import com.comverse.rtbd.security.sdk.policy.client.PDPClient;
import com.comverse.rtbd.security.sdk.policy.client.RemotePDPClient;
import com.comverse.rtbd.security.sdk.policy.common.PolicyUtils;
import com.sun.xacml.ctx.RequestCtx;
import com.sun.xacml.ctx.Result;

public class TestAuthorization {

    /**
     * @param args
     */
    public static void main(String[] args) {

        try {
            System.setProperty("security.server.ip", "10.230.20.91");
            System.setProperty("java.security.auth.login.config", "./conf/sec_jaas.conf");
            System.setProperty("javax.net.ssl.trustStore", "./conf/upsec.keystore");
            System.setProperty("javax.net.ssl.trustStore-pwd", "security");
            System.setProperty("security.client.context", "Client_Security");
            System.setProperty("security.webservice.stack", "weblogic");
            System.setProperty("bea.home", "./conf");
            System.setProperty("weblogic.webservice.client.ssl.strictcertchecking", "false");
            System.setProperty("weblogic.webservice.client.ssl.adapterclass",
                    "weblogic.webservice.client.JSSEAdapter");

            PDPClient pdp = new EmbeddedPDPClient("/policy", "CSM_CSM*", null);
```

Figure 49 Policy Management Code Example, Part 2

```
            if (ConfigUtils.importSecurityCertificate()) {
                System.out.println("The certificate created");
            } else {
                System.out.println("The certificate creation failed");
            }

            LoginContext context = new LoginContext("csmuser", "CSM",  "passw0rd");
            context.login();

            // Retrieve Token
            String tokenStr = context.getSSOToken();

            for (int i=0; i<1; i++) {
                int retVal = pdp.isAuthorized(tokenStr, "APPLICATION_RSRC:FXLauncher", "READ");
                System.out.println("The retval = " + retVal);
                if (retVal == Result.DECISION_PERMIT) {
                    System.out.println("FXLauncher is authorized to the user");
                } else if (retVal == Constants.TOKEN_EXPIRED){
                    System.out.println("Token expired");
                } else if (retVal == Result.DECISION_NOT_APPLICABLE) {
                    System.out.println("Decision not applicable");
                } else if (retVal == Result.DECISION_INDETERMINATE) {
                    System.out.println("Decision indeterminate");
                } else if (retVal == Result.DECISION_DENY) {
                    System.out.println("FXLauncher is not authorized for the user");
                }
            }

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }


    }

}
```

Audit Management API Code Example

The code example in Figure 50 and Figure 51 on page 99 demonstrates how client application can initiate an audit session, create audit records, and send (commit) audit records to the Unified Platform Agent (UPA). An audit session is established by creating an instance of the `XDasSession` class. After the session is established, client applications can generate any number of audit records. *This code example is for demonstration purposes only. It does not represent actual code to be used.*

Figure 50 Auditing Code Example, Part 1

```
package com.comverse.rtbd.security.sdk.test.adt;

import java.io.IOException;

import com.comverse.rtbd.security.sdk.audit.xdas.XDasEvents;
import com.comverse.rtbd.security.sdk.audit.xdas.XDasException;
import com.comverse.rtbd.security.sdk.audit.xdas.XDasOutcomes;
import com.comverse.rtbd.security.sdk.audit.xdas.XDasRecord;
import com.comverse.rtbd.security.sdk.audit.xdas.XDasSession;

/**
 * Class for testing audit API implementation
 *
 */
public class XdasTest {
    public static void main(String[] args)  {

        System.setProperty("security.sdk.mode", "false");

        //optional property....defaults to local host
        //System.setProperty("sdk.audit.host", "10.150.1.144");

        System.out.println("Executing XdasTest");

        try {

            // creating a Xdas session
            XDasSession session = new XDasSession("PaymentDatabase");


            XDasRecord rec = session.XDasStartRecord(0,0,"JavaInitAuth:JavaDomainName:JavaDomainId_java3",
                    null,"Test=Yes,Debug=exception,Meaningless=Exception");

            //set outcome
            rec.setOutcome(XDasOutcomes.XDAS_OUT_ACTIONS_SET);

            rec.setTargetInfo("JavaTargetLocName", "Locationadress", "TargetService",
                    "TargetAuthAuthority", "TargetPrincipalName", "TargetPrincipalIdentity");
            rec.setInitiatorInfo("AuthAuthority", "DomainName", "DomainSpecificId");
```

Figure 51 Auditing Code Example, Part 2

```
            //set event number
            rec.setEventNumber(XDasEvents.XDAS_AE_CREATE_ACCOUNT);
            //set external id
            rec.setExternalId("12345");

            //commit record
            rec.commit();

            //create another record using same session
            XDasRecord rec1=session.XDasStartRecord(XDasEvents.XDAS_AE_CREATE_ACCOUNT,
                    XDasOutcomes.XDAS_OUT_ACTIONS_SET,"JavaInitAuth:JavaDomainName:JavaDomainId_java3",
                            "JavaTargetLocName:JavaTargetLocAddr:JavaTargetService:JavaTargetAuth:
                                    JavaTargetPrincipal:JavaTargetPrincipalId",
                            "Test=Yes,Debug=exception,Meaningless=Exception");
            rec1.commit();

        } catch (XDasException ex) {
            System.out.println("XDas Error: " + ex.getStatus()  + ", Minor Code: " + ex.getMinorStatus());
        } catch (IOException e) {
            e.printStackTrace();
        }


    }
}
```

Credentials Management API Code Examples

These code examples are for demonstration purposes only. They do not represent actual code to be used.

Database Passwords

The code example in Figure 52 demonstrates how database passwords can be fetched from the Security Server using static methods defined in the `DBPasswordManagement` class.

Figure 52 Fetch Database Password Code Example

```
package com.comverse.rtbd.security.sdk.test;

import com.comverse.rtbd.security.sdk.dbpwd.DBPasswordManagement;
import com.comverse.rtbd.security.sdk.exceptions.DBPwdException;

public class DBPwdTest
{
    static String pwd= null;

    public static void main(String[] args)
    {

        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.230.16.208");

        System.out.println("Ececuting DBPwd Test");
        try {

                pwd = DBPasswordManagement.getDBPassword("CBS", "MAIN", "user1",false);
                System.out.println("Password fetched from security server = "+ pwd);

                pwd = DBPasswordManagement.getDBPassword("CBS", "HIST", "user2",true);
                System.out.println("Password fetched from local cache = "+ pwd);

        } catch (DBPwdException e) {
                e.printStackTrace();
        }


    }
}
```

SNMP Community Strings

The code example in Figure 53 demonstrates how SNMP community strings can be fetched from the Security Server using static methods defined in the `CredentialReqHandler` class.

Figure 53 Fetch SNMP Community String Code Example

```
package com.comverse.rtbd.security.sdk.test.credential;

import com.comverse.rtbd.security.sdk.credential.CredentialReqHandler;
import com.comverse.rtbd.security.sdk.exceptions.CredentialException;

public class CredTest
{


    public static void main(String[] args)
    {
        String cs= null;

        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.150.1.143");
        System.setProperty("credential.secure.mode", "false");

        System.out.println("Executing Credential Test");

        try {
            cs=CredentialReqHandler.getSNMPCommString("cajun", "cajun", "inst",false);
            System.out.println("Community string : "+ cs);
            } catch (CredentialException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
}
```

Key Management API Code Example

This code example is for demonstration purposes only. It does not represent actual code to be used.

The code example in Figure 54 and Figure 55 on page 103 demonstrates how client applications can use the Key Management API to create keys, retrieve symmetric keys, and disable symmetric keys. In the example, the circled area indicates an optional system property.

Figure 54 Symmetric Keys Code Example, Part 1

```
package com.comverse.rtbd.security.sdk.test;

import java.security.PublicKey;
import com.comverse.rtbd.security.sdk.exceptions.KeyManagerClientException;
import com.comverse.rtbd.security.sdk.key.KeyManagerClient;
import com.comverse.rtbd.security.sdk.key.SymKey;


/**
 *  class for testing KeyManagementClient.
 *
 */
public class KeyManagementClientTest
{
    static KeyManagerClient kmc=null;

    public static void main(String[] args)
    {

        SymKey aesKey=null;
        SymKey blowhishKey=null;
        String keyId ="01-234859";

        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.73.17.114");
        System.setProperty("sdk.keystore.path", "./conf");


        System.out.println("Executing KeyManagementClientTest");
    try {
         kmc= KeyManagerClient.getInstance();
    } catch (KeyManagerClientException e) {
            e.printStackTrace();
    }

    System.out.println("fetching Security Server Public Key");
    try {
            PublicKey pk = kmc.getSecurityServerPublicKey();
            System.out.println("Security Server Public Key is fetched");

        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }



      System.out.println("generating blowfish secret Key");
        try {
            blowhishKey=kmc.generateKey("Blowfish");

            System.out.println("Symkey mmemebrs...");
            System.out.println(blowhishKey.getAlgorithm());
            System.out.println(blowhishKey.getStatus());
            System.out.println(blowhishKey.getKeyId());
            System.out.println(blowhishKey.getkey());

        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }
```

Figure 55 Symmetric Keys Code Example, Part 2

```
        System.out.println("generating AES secret Key with user defined key Id");

        try {
            aesKey=kmc.generateKey("AES", "01-12345");
            System.out.println("AES secret Key generated");
            System.out.println("Symkey mmemebrs...");
            System.out.println(aesKey.getAlgorithm());
            System.out.println(aesKey.getStatus());
            System.out.println(aesKey.getKeyId());
            System.out.println(aesKey.getkey());

        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }


        System.out.println("fetching blowfish key using key Id");

        try {
            kmc.getKey(keyId);
            System.out.println(" blowfish key fetched");

        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }




        System.out.println("disabling blowfish key using KeyId");

        try {
            if(kmc.disableKey("01-12345"))
            {
                System.out.println("Key disabled");
            }

        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }

        System.out.println("disabling  blowfish key which is already inactive");
        try {
            System.out.println("result :"+ kmc.disableKey(keyId));
        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }

        System.out.println("trying to fetch blowfish key which is disabled");
        try {
            kmc.getKey(keyId);
        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }


    }
}
```

Cryptography API Code Examples

This section contains code examples related to symmetric-key and password-based encryption, digital signatures, and message digests. *These code examples are for demonstration purposes only. They do not represent actual code to be used.*

Symmetric Key Encryption

The code example in Figure 56 and Figure 57 on page 105 demonstrates how client applications can encrypt and decrypt data using `AESEncryptor`. The example shows how to fetch the symmetric key from the Security Server.

Figure 56 AES Encryption/Decryption Code Example, Part 1

```
package com.comverse.rtbd.security.sdk.test.aes;

import javax.crypto.SecretKey;

import com.comverse.rtbd.security.sdk.crypto.EncryptorProvider;
import com.comverse.rtbd.security.sdk.crypto.SymmetricEncryptor;
import com.comverse.rtbd.security.sdk.exceptions.EncryptorException;
import com.comverse.rtbd.security.sdk.exceptions.KeyManagerClientException;
import com.comverse.rtbd.security.sdk.key.KeyManagerClient;
import com.comverse.rtbd.security.sdk.key.SymKey;

public class StringEncTest
{
     //method to encrypt data.
    public String encrypt(String str ,SecretKey key)
    {

        SymmetricEncryptor be=EncryptorProvider.createAESEncryptor();
        be.setKey(key);
        try {
            return be.encrypt(str);
        } catch (EncryptorException  e) {
                e.printStackTrace();
        }

        return null;
    }

      //method to decrypt data.
    public String decrypt(String str ,SecretKey key)
    {
        SymmetricEncryptor be=EncryptorProvider.createAESEncryptor();
        be.setKey(key);
        try {
            return be.decrypt(str);
        } catch (EncryptorException  e) {
                    e.printStackTrace();
        }

        return null;
    }
```

Figure 57 AES Encryption/Decryption Code Example, Part 2

```
    public static void main(String[] args)throws Exception
    {
        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.150.1.144");


        KeyManagerClient kc =null;
        try {
            kc = KeyManagerClient.getInstance();
        } catch (KeyManagerClientException e1) {
                e1.printStackTrace();
        }

        SymKey sk =null;

        try {
            sk = kc.getKey("PCI_DB_FLD");
        } catch (KeyManagerClientException e) {
                e.printStackTrace();
        }
        SecretKey key = sk.getkey();


        System.out.println("Executing AES StringTest");
        StringEncTest bt =new StringEncTest();

        String enc_str=bt.encrypt("comverse test string", key);

        System.out.println("encrypted String :"+enc_str);
        System.out.println("Decrypted String :" + bt.decrypt(enc_str, key));
    }

}
```

The code example in Figure 58 and Figure 59 on page 107 demonstrates how client applications can encrypt and decrypt a file using `BlowfishEncryptor`. The example shows how to fetch the symmetric key from the Security Server.

Figure 58 Blowfish Encryption/Decryption Code Example, Part 1

```
package com.comverse.rtbd.security.sdk.test.Blowfish;

import javax.crypto.SecretKey;

import com.comverse.rtbd.security.sdk.crypto.EncryptorProvider;
import com.comverse.rtbd.security.sdk.crypto.SymmetricEncryptor;
import com.comverse.rtbd.security.sdk.exceptions.EncryptorException;
import com.comverse.rtbd.security.sdk.exceptions.KeyManagerClientException;
import com.comverse.rtbd.security.sdk.key.KeyManagerClient;
import com.comverse.rtbd.security.sdk.key.SymKey;

public class FileTest
{
       //Encrypting data
    public void encrypt(String inp,String out ,SecretKey key)
    {

        SymmetricEncryptor be=EncryptorProvider.createBlowfishEncryptor();
        be.setKey(key);
        try {
            be.encrypt(inp,out);
        } catch (EncryptorException  e) {
                e.printStackTrace();
        }


    }

      //Decrypting data
    public void decrypt(String inp,String out ,SecretKey key)
    {
        SymmetricEncryptor be=EncryptorProvider.createBlowfishEncryptor();
        be.setKey(key);
        try {
             be.decrypt(inp,out);
        } catch (EncryptorException  e) {
                    e.printStackTrace();
        }


    }
```

Figure 59 Blowfish Encryption/Decryption Code Example, Part 2

```
    public static void main(String[] args)throws Exception
    {
        System.setProperty("security.sdk.mode", "false");
        System.setProperty("security.server.ip", "10.150.1.144");


        KeyManagerClient kc =null;
        try {
            kc = KeyManagerClient.getInstance();
        } catch (KeyManagerClientException e1) {
                e1.printStackTrace();
        }

        SymKey sk =null;
        try {
            sk = kc.getKey("PCI_PMT_COM");
        } catch (KeyManagerClientException e) {
                e.printStackTrace();
        }
        SecretKey key = sk.getkey();

        FileTest bt =new FileTest();

        System.out.println("Executing Blowfish Test");

        bt.encrypt("inp.txt","res.txt", key);
        bt.decrypt("out.txt","rec.txt", key);


    }
}
```

Password-Based Encryption

The code example in Figure 60 demonstrates how client applications can perform password-based encryption/decryption using `StandardPBEStringEncryptor`.

Figure 60 Password-Based Encryption/Decryption Example

```
package com.comverse.rtbd.security.sdk.test;

import java.io.BufferedReader;
import java.io.InputStreamReader;

import org.jasypt.encryption.pbe.StandardPBEStringEncryptor;


//testing password based encryption
public class PBEStringTest
{
       //Encrypting data
    public String encrypt(String data ,String password)
    {
        StandardPBEStringEncryptor te=new StandardPBEStringEncryptor ();
        te.setPassword(password);
        return te.encrypt(data);
    }

      //Decrypting data
    public String decrypt(String  enc_data ,String password)
    {
        StandardPBEStringEncryptor te=new StandardPBEStringEncryptor ();
        te.setPassword(password);
        return te.decrypt(enc_data);
    }

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

        PBEStringTest pt =new PBEStringTest();

        BufferedReader stdin = new BufferedReader(
                new InputStreamReader( System.in ) );

        System.out.println("Executing PBEStringTest");
        System.out.println("Enter the string to be encrypted :");
        String plain=stdin.readLine();

        String enc_str = pt.encrypt(plain,"test");

        System.out.println("plain text :"+ plain);
        System.out.println("encrypted text :"+ enc_str);
        System.out.println("decrypted text :"+ pt.decrypt(enc_str,"test"));


    }
}
```

Digital Signatures

The code example in Figure 61 and Figure 62 on page 110 demonstrates how client application can use `SignatureProvider` to digitally sign data and verify digital signatures.

Figure 61 Digital Signature Code Example, Part 1

```
package com.comverse.rtbd.security.sdk.test;

import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;
import com.comverse.rtbd.security.sdk.crypto.signature.SignatureProvider;
import com.comverse.rtbd.security.sdk.exceptions.KeyManagerClientException;
import com.comverse.rtbd.security.sdk.exceptions.SignatureProviderException;
import com.comverse.rtbd.security.sdk.key.KeyManagerClient;

//test class for signatureprovider
public class SignatureTest
{
    SignatureProvider sp;
    boolean result;
    byte[] signature;

    SignatureTest()
    {
        sp=null;
        result=false;
        signature=null;
    }

    //signing of data
    public byte[] signdata(byte[] data ,PrivateKey prkey)
    {
        sp = new SignatureProvider();
        sp.setPrivateKey(prkey);
         try {
            signature=sp.sign(data);
        } catch (SignatureProviderException e) {
                e.printStackTrace();
        }
        return signature;
    }

      //verifying signature
    public boolean verifydata(byte[] data ,byte[] signature,PublicKey pukey)
    {
     sp=new SignatureProvider();
        sp.setPublicKey(pukey);

        try {
            result = sp.verify(data, signature);
        } catch (SignatureProviderException e) {
            e.printStackTrace();
        }
        return result;
    }
```

Figure 62 Digital Signature Code Example, Part 2

```
    public static void main(String[] args)
    {
        System.setProperty("security.sdk.mode", "true");
        System.setProperty("security.server.ip", "localhost");;

        SignatureTest st =new SignatureTest();
        byte[] data = "abcd".getBytes();

        //fetch keys from KMC
        KeyManagerClient kmc=null;
        try {
            kmc = KeyManagerClient.getInstance();
        } catch (KeyManagerClientException e1) {
                e1.printStackTrace();
        }
        KeyPair kp=null;
        try {
            kp = kmc.generateRSAKeyPair();
        } catch (KeyManagerClientException e) {
            e.printStackTrace();
        }

        PublicKey puk = kp.getPublic();
        PrivateKey prk = kp.getPrivate();

        byte[] s=st.signdata(data,prk);

        System.out.println("Executing SignatueTest");
        System.out.println("Signature Result ="+st.verifydata(data, s, puk));

    }
}
```

Message Digests

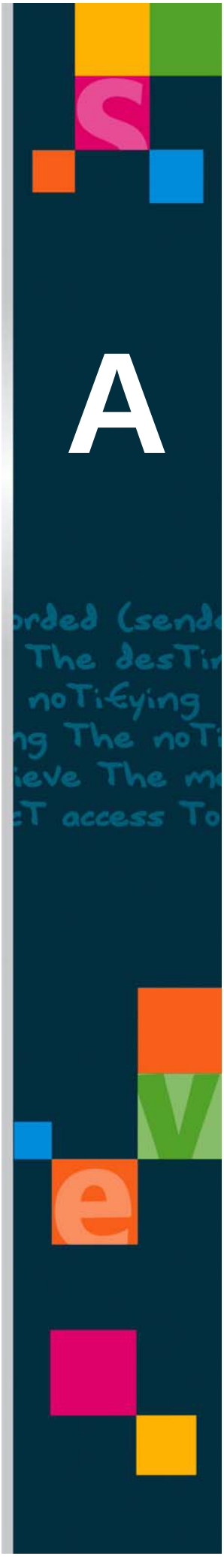
The code example in Figure 63 demonstrates how client application can use `StandardStringDigester` for message digests. The example shows how to create the message digest and verify the signature.

Figure 63 Message Digest Code Example

```
package com.comverse.rtbd.security.sdk.test;

import com.comverse.rtbd.security.sdk.crypto.digest.StandardStringDigester;


public class StringDigestTest
{
    //create digest
    public String createDigest(String str)
    {
        StandardStringDigester sd =new StandardStringDigester();
        sd.initialize();
        return sd.digest(str);

    }

    //verify digest
    public boolean verifyDigest(String str, String digest)
    {
        StandardStringDigester sd =new StandardStringDigester();
        sd.initialize();
        boolean result=sd.matches(str, digest);
        return result;
    }

    public static void main(String[] args)
    {

        StringDigestTest dt = new StringDigestTest();
        String dig = dt.createDigest("abcd");

        System.out.println("Executing StringDigestTest");
        System.out.println(dt.verifyDigest("abcd", dig));

    }

}
```

Appendix A

Attribute Conflict-Resolution Rules



Overview

Custom attributes can be defined at the security realm, security realm group, and user levels. Note the following points about attributes:

- All users provisioned for the realm, regardless of their group membership, inherit the realm's attributes.
- Users also inherit the attributes for all groups to which they belong.
- Individual users can also have custom attributes defined.

Because of this multiple-level attribute definition and inheritance scheme, the potential exists for conflicts among attributes. For example, suppose an attribute with the same name is defined at the realm, group, and user levels but the attribute is assigned a different value at each level. The result is an attribute conflict when determining which attribute value to use. This appendix describes the conflict-resolution rules used to resolve attribute conflicts and provides examples illustrating the rules.

Conflict-Resolution Rules

The general solution is that a user attribute overrides a group attribute, which overrides a realm attribute. Because the situation can be more complex than that general solution addresses, the following are the rules for resolving attribute conflicts:

At the *realm* level, no conflict-resolution rules exist because this is the top level.

At the *group* level, the rules are processed in the following order:

1. If a conflict exists between attributes for the priority group and a secondary group, the priority group wins.
2. If a conflict exists between attributes for secondary groups, a random selection is performed to determine which group wins. (It is expected that this conflict will be resolved at the user level. Stated another way, attributes must be defined at the user level to resolve the conflict.)
3. If a conflict exists between attributes at the realm level and group level, the group level wins. (This determination is performed last, once all group resolution rules have been performed.)

At the *user* level, if a conflict exists between attributes at the group level and the user level, the user level wins.

Examples

The examples in this section provide different scenarios to illustrate the attribute conflict-resolution rules.

Scenario 1

GROUP A (priority group defined for the USER)

Attribute: A1 = 2

GROUP B (secondary group defined for the USER)

Attribute: A1 = 4

USER

Attribute: A1 = 5

In this scenario, the USER belongs to both GROUP A and GROUP B. Each of the groups has an attribute named A1 but with different values for the attribute. The USER also has an attribute named A1, with still another value for the attribute.

Between the groups, GROUP A (A1 = 2) wins because it is the priority group. However, in this scenario, the user attribute (A1 = 5) ultimately wins because user attributes have a higher priority than group attributes.

Scenario 2

GROUP A (priority group defined for the USER)

Attribute: No A1 attribute defined

GROUP B (secondary group defined for the USER)

Attribute: A1 = 4

USER

Attribute: A1 = 5

In this scenario, the USER belongs to both GROUP A and GROUP B. The priority group has no attribute named A1, but a secondary group has an attribute named A1. The USER also has an attribute named A1, but with a different value for the attribute.

In this scenario, the user attribute (A1 = 5) wins because user attributes have a higher priority than group attributes.

Scenario 3

GROUP A (priority group defined for the USER)

Attribute: No A1 attribute defined

GROUP B (secondary group defined for the USER)

Attribute: A1 = 4

GROUP C (secondary group defined for the USER)

Attribute: A1 = 6

USER

Attribute: No A1 attribute defined

In this scenario, the USER belongs to GROUP A, GROUP B, and GROUP C. The priority group has no attribute named A1, and neither does the USER. Between the secondary groups with conflicting attributes named A1, the winner is determined randomly (that is, the winning attribute will be either A1 = 4 or A1 = 6).

In this scenario, the conflict must be resolved by definition of the A1 attribute and value at the user level.

Scenario 4

REALM

Attribute: A1 = 1

GROUP A (priority group defined for the USER)

Attribute: A1 = 2

GROUP B (secondary group defined for the USER)

Attribute: A1 = 4

USER

Attribute: A1 = 5

In this scenario, the REALM has an attributed named A1 defined. The USER belongs to both GROUP A and GROUP B, and each of the groups has an attribute named A1. The USER also has an attribute named A1. Values for the A1 attribute are different for the realm, both groups, and the user.

Between the groups, GROUP A (A1 = 2) wins because it is the priority group. However, in this scenario, the user attribute (A1 = 5) ultimately wins because user attributes have a higher priority than both group attributes and realm attributes.

Index

T

U

X