SYBASE®
An SAP® Company

Developer Guide: Unwired Server Management API

Sybase Unwired Platform 2.1

DOCUMENT ID: DC01332-01-0210-01
LAST REVISED: October 2011

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

Introducing Developer Guide for Unwired Server Management API

This guide provides information about using the Sybase® Unwired Platform Administration APIs to custom code an administration client. The audience is advanced developers who are familiar working with APIs, but who may be new to Sybase Unwired Platform.

This guide describes the features and usage of the Administration API, how to get started with client development, and how to program a custom administration client. Also included is information on how to configure Unwired Platform properties using client metadata, how to use properties, and a listing of error codes.

Administration Client API Features

Sybase Unwired Platform includes a Java API that opens the administration and configuration of Sybase Unwired Platform to Java client applications you create. By building a custom client with the administration client API, you can build custom application to support Sybase Unwired Platform administration features and functionality within an existing IT management infrastructure.

When creating a custom Unwired Platform administration client, the entry point is the `SUPObjectFactory` class. By calling methods of `SUPObjectFactory`, which require different context objects, you can retrieve administration interfaces to perform administration activities. Should errors occur, they are reported through a `SUPAdminException`, which provides the error code and error message. For details of each administration interface, you can refer to the Javadoc shipped with the administration client API.

Companion Docs

Companion guides include:

- *System Administration*
- *Sybase Control Center for Sybase Unwired Platform*
- *Sybase Unwired WorkSpace – Mobile Business Object*
- *Troubleshooting for Sybase Unwired Platform*

See *Fundamentals* for high-level mobile computing concepts, and a description of how Sybase Unwired Platform implements the concepts in your enterprise.

Javadocs

The administration client API installation includes Javadocs. Use the Sybase Javadocs for your complete API reference.

As you review the contents of this document, ensure you review the reference details documented in the Javadoc delivered with this API. By default, Javadocs are installed to `<UnwiredPlatform_InstallDir>\Servers\UnwiredServer\AdminClientAPI\com.sybase.sup.adminapi\docs\api\index.html`.

The top left navigation pane lists all packages installed with Unwired Platform. The applicable documentation is available with `com.sybase.sup.admin.client` package. Click this link and navigate through the Javadoc as required.

Documentation Roadmap for Unwired Platform

Learn more about Sybase® Unwired Platform documentation.

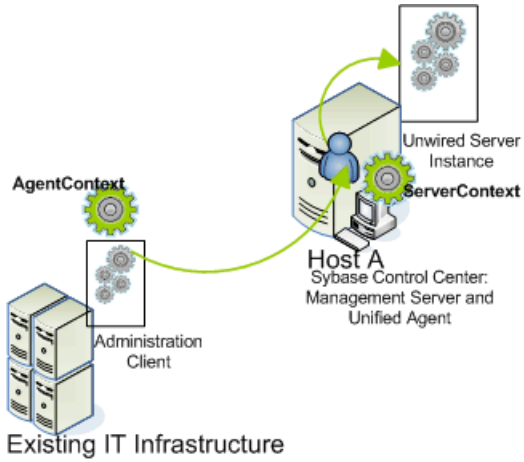
See *Documentation Roadmap* in *Fundamentals* for document descriptions by user role. *Fundamentals* is available on Production Documentation.

Check the Sybase Product Documentation Web site regularly for updates: access *http://sybooks.sybase.com/nav/summary.do?prod=1289*, then navigate to the most current version.

Administration Client API

The client you create connects to Unwired Server through Sybase Control Center and Sybase Unified Agent.

For example, as this illustration shows, connections are established using an `AgentContext` and a `ServerContext`:



You do not need to create an instance of `AgentContext`. If none is defined, a default one is created by the `ServerContext` using its host value and default agent port (9999).

Contexts

A context is a lightweight, immutable object that is used to retrieve a specific administration interface instance. You create a connection to the Unwired Server when you invoke an API (such as `ping`) on a supported interface (such as `SUPServer`), but not when context objects (such as `AgentContext` or `ServerContext`) are initialized. There is no need to maintain the states of contexts because state changes are not supported.

The administration client API includes these contexts:

Context	Description
`AgentContext`	Optional. Connects to the Unified Agent that acts as a proxy and manages the connection to the Unwired Server instance identified in the `ServerContext`.

Context	Description
`DefaultAdmin-Context`	The super class of other concrete context classes.
`AdminContext`	The `AdminContext` is an interface that all context classes implement.
`ServerContext`	Required to connect to the Unwired Server instance. If you don't specify an `AgentContext`, the `ServerContext` creates one for you using default values. See *Connecting to an Unwired Server Instance*. Use this context to retrieve the `ClusterContext` .
`ClusterContext`	Required to manage a specific cluster. Use this context to retrieve the `DomainContext`.
`DomainContext`	Required to manage a specific domain. Use this context to retrieve the `PackageContext`
`PackageContext`	Required to deploy and manage a package. Use this context to retrieve the `MBOContext`
`MBOContext`	Required to manage a mobile business object. Use this context to retrieve the `OperationContext`
`OperationCon-text`	Required to manage an operation.
`SecurityContext`	Required to manage the security for the platform

For details on these classes, and the methods that implement them, see the Javadocs for `com.sybase.sup.admin.client`.

See also

- *Connecting to an Unwired Server Instance* on page 8

Administration Interfaces

The administration client API uses several interfaces that contain operations which can be invoked by custom code to perform management of the Unwired Server.

The administration client API includes these administration interfaces:

Interface	Includes methods that
`SUPServer`	Command and control operations for an Unwired Server instance, for example start, stop, and ping.

Interface	Includes methods that
`SUPCluster`	Manage cluster security, monitoring configuration and domain creation for a cluster instance, and so on.
`SUPDomain`	Manage domains, deploy packages to a domain, set security configurations for a domain, and so on.
`SUPPackage`	Configure packages by setting up subscriptions, configuring cache groups, configuring endpoint properties, and so on.
`SUPMobileBusinessObject`	View mobile business object properties, operations, errors, endpoints, and so on.
`SUPOperations`	View operation properties, errors, endpoints, and so on.
`SUPApplication`	Manage applications, application connections, and application connection templates
`SUPMonitor`	Perform monitoring functions like viewing histories, summaries, details, and performance data for various platform components, and export data as required.
`SUPServerLog`	View, filter, delete and refresh logs, configure appenders, and so on, for Unwired Server and its embedded services like replication and messaging synchronization.
`SUPDomainLog`	Configure domain log settings and view, filter, delete domain logs entries, and so on.
`SUPServerConfiguration`	Configure an Unwired Server instance, as well as its listeners. All methods of this interface, except the apple push notification-related properties are metadata-based.
`SUPSecurityConfiguration`	Create, manage, and configure a security configuration with at least one authentication provider. You can add other providers (authentication, authorization, attribution, and audit) as required.
`SUPMobileWorkflow`	Manage and configure deployed mobile workflow packages.

For details on these classes, and the methods that implement them, see the Javadocs for `com.sybase.sup.admin.client`.

See also

- *Client Metadata* on page 165

SUPObjectFactory

Once a context has been instantiated, pass it to a specific method of `SUPObjectFactory` to retrieve an administration interface. You can then start administration by calling methods of the interface.

The methods in the `SUPObjectFactory` class can accept an instance of `AdminContext` as a parameter. For example, to get an administration interface of `SUPServer`, you must create an instance of `ServerContext` with the correct information and pass it to `SUPObjectFactory.getSUPServer()`.

`SUPObjectFactory` provides a `shutdown()` method to cleanly shut down an application that uses the API. See the Javadocs for details.

Metadata

Metadata-based configuration is used by these administration components:

- Unwired Server configuration properties
- Unwired Server log configuration properties
- Security configurations and the providers used in those configurations
- Endpoint connection properties

See also

- *Client Metadata* on page 165

Exceptions and Error Codes

The administration client API throws only one checked exception, `SUPAdminException`.

An error code is associated with each thrown `SUPAdminException`, so that developers can easily diagnose what happened when the exception is thrown.

Note: See *Developer Guide for Unwired Server Management API > Error Code Reference* for a list of predefined error codes.

Getting Started with Client Development

An Unwired Platform development cycle includes several steps.

1. *Required Files*

   The following files are required in your class path.
2. *Starting Required Services*

   Before beginning development, you must start required Unwired Platform services so you can connect to them.
3. *Connecting to an Unwired Server Instance*

   AgentContext and ServerContext are lightweight, immutable Java objects.
4. *Developing Client Contexts, Objects, and Operations*

   Once you have an instance of ServerContext, you can create other contexts from it.

Prerequisites

Review this list to understand what prerequisites to consider before starting the development of a custom administration tool within an existing enterprise-level adminstration framework.

- A development environment that supports Java development, for example, Eclipse.
- Optionally, if you want to install Sybase Control Center, it must be installed on the same host as Unwired Server.

Required Files

The following files are required in your class path.

- sup-admin-pub-client.jar
- sup-admin-pub-common.jar
- castor-1.2.jar
- commons-beanutils-core-1.7.0.jar
- commons-lang-2.2.jar
- commons-logging-1.1.1.jar
- commons-pool-1.4.jar
- sup-at-lite.jar
- sup-mms-admin-api-lite.jar
- uaf-client.jar

- log4j-1.2.6.jar
- commons-codec-1.3.jar
- log4j.properties (the file residing in the sample folder can be a template)

By default, the sup-admin-pub-client.jar, and sup-admin-pub-common.jar files are installed to the `<UnwiredPlatform_InstallDir>\Servers\UnwiredServer\AdminClientAPI\com.sybase.sup.adminapi` folder. All other jar files can be found in the `<UnwiredPlatform_InstallDir>\Servers\UnwiredServer\AdminClientAPI\com.sybase.sup.adminapi\lib` folder.

Note: If you have Xerces J-Parser installed and have xerces.jar (the parser class files) in your class path, the xerces.jar library may cause a class conflict with Sybase Unwired Platform. This problem only occurs in certain circumstances when JDK 6 is used with Xerces. If this problem occurs, you must remove this jar from your class path.

Starting Required Services

Before beginning development, you must start required Unwired Platform services so you can connect to them.

Prerequisites

Ensure the required service are all installed on the same host.

Task

By starting required services, you start the servers and dependent services. For a complete list of Unwired Platform Services, see *System Administration > System Reference > Unwired Platform Windows Services*.

1. Click the **Start Unwired Platform Services** desktop shortcut to start Unwired Server and the dependent services that the custom tool you develop will manage.
2. Use the Services Control Panel to verify that the Windows service named **Sybase Control Center X.X** is started. If it has not, start it by selecting the service and clicking **Start**.

Connecting to an Unwired Server Instance

`AgentContext` and `ServerContext` are lightweight, immutable Java objects.

Creating either of these objects does not immediately establish a connection to either Sybase Control Center or the Unwired Server.

1. (Optional) Create an `AgentContext` object.

The default constructor creates an instance with host="localhost", port="9999", user="" and password="". The constructor in this sample creates an instance with host="<host name>", port="9999", user="supAdmin" and password="s3pAdmin":

```
AgentContext agentContext = new AgentContext();
agentContext = new AgentContext("<host name>", 9999, "supAdmin",
"s3pAdmin");
```

2. Create a `ServerContext` object.

   Every `ServerContext` instance has an `AgentContext` instance. When you instantiate `ServerContext`, you can pass an instance of `AgentContext` to the constructor. If you do not specify an `AgentContext`, the constructor automatically creates an `AgentContext` with the same host, user name, and password values as those defined in the `ServerContext`.

   It also assigns 9999 as the port number for `AgentContext`, for these reasons:

   - Unwired Server and Sybase Control Center are installed on the same host, and they share the same security provider.
   - By default, Sybase Control Center listens on port 9999. The administration API connects to Sybase Control Center using this port.

   This sample creates a `ServerContext` that uses values of supAdmin and s3pAdmin for the user name and password, and uses secure port (2001) by specifying "true" in the last parameter:

   ```
   ServerContext serverContext = new ServerContext();
   serverContext = new ServerContext("<host name>", 2001, "supAdmin",
   "s3pAdmin", true);
   ```

   The usage of secure port does not require server certificate installation on the client-side. It is assumed that server is configured with a valid and secure certificate for transport level security, and client authentication is done via the security provider assigned to the 'admin' security configuration.

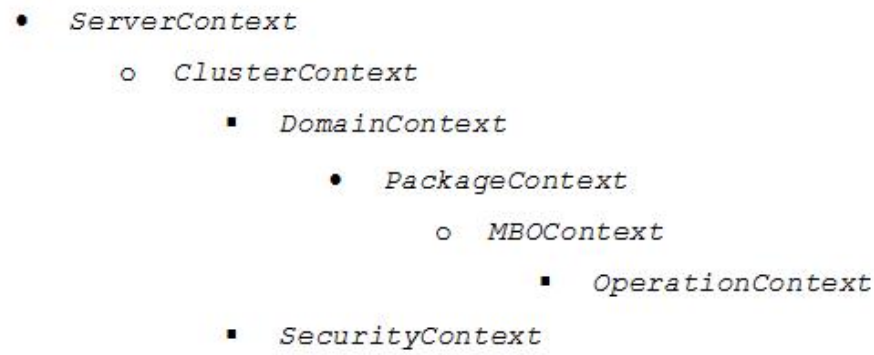
See also

- *Contexts* on page 3

Developing Client Contexts, Objects, and Operations

Once you have an instance of `ServerContext`, you can create other contexts from it.

1. Create required client artifacts.
   - Create the context objects you require. The following diagram illustrates the subclasses of `AdminContext` and their logical hierarchy.



The following code fragment creates multiple contexts for cluster, security, domain, package, mobile business objects, and operations:

```
ClusterContext clusterContext =
serverContext.getClusterContext("<cluster name>");
SecurityContext securityContext =
clusterContext.getSecurityContext("<security configuration
name>");
DomainContext domainContext =
clusterContext.getDomainContext("<domain name>");
PackageContext packageContext =
domainContext.getPackageContext("<package name>");
MBOContext mboContext = packageContext.getMBOContext("<MBO
name>");
OperationContext operationContext =
mboContext.getOperationContext("<operation name>");
```

- Call methods of `SUPObjectFactory` to create the administration interface required. For example, to create an object of `SUPServer`, pass an instance of `ServerContext` to `SUPObjectFactory` by calling:

  ```
  SUPObjectFactory.getSUPServer(serverContext);
  ```

2. Once the administration session ends, clean the resources held by the API by calling `SUPObjectFactory.shutdown()`. This method is provided only to help your administration application exit cleanly, and is not designed to be called after each administration operation.

   For example:

   ```
   SUPObjectFactory.shutdown();
   ```

3. Build the client application.

Code Samples

Use the Javadocs for the administration client API package with the interface code samples to understand how to program a custom administration client.

Code samples are organized by the interface used.

Controlling Unwired Server (SUPServer Interface)

The `SUPServer` interface allows you to manage the Unwired Server.

Operations you can perform with this interface include:

- Starting an administration session for an Unwired Server instance.
- Retrieving Unwired Server properties and status.
- Performing command and control actions like starting and stopping.

Session Start-up

Starts the management of an Unwired Server instance.

Syntax

```
public static SUPServer getSUPServer(ServerContext serverContext)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Session Start-up –**

  ```
  SUPServer supServer =
  SUPObjectFactory.getSUPServer(serverContext);
  ```

Usage

When an instance of `SUPServer` is returned from the SUPObjectFactory, call its method. The state of the connection to the Unwired Server is automatically managed; an explicit connection to the Unwired Server is not required.

Server Properties Retrieval

Retrieves the general properties of the Unwired Server instance.

Syntax

```
ServerVO getProperties() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Getting properties –** gets the properties for a server instance named ServerVO:
  ```
  ServerVO svo = supServer.getProperties();
  ```

Status Verification

Checks if the Unwired Server instance is available.

Syntax

```
void ping() throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Ping –** pings an Unwired Server to see if it is available:
  ```
  supServer.ping();
  ```

Server Start-up

Starts an Unwired Server instance.

Syntax

```
void start() throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Startup –**
  ```
  supServer.start();
  ```

Server Shutdown

Stops an Unwired Server instance.

Syntax

```
void stop() throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Shutdown –**
  ```
  supServer.stop();
  ```

Server Restart

Restarts an Unwired Server instance.

Syntax

```
void restart() throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Restart –**
  ```
  supServer.restart();
  ```

Managing Clusters

The `SUPCluster` interface allows you manage the cluster to which the Unwired Server instance belongs.

Operations you can perform with this interface include:

- Listing member servers, suspending/resuming member servers
- Listing, creating, and deleting domains
- Listing, creating, and deleting security configurations

- Listing, creating, updating, and deleting monitoring configurations, deleting monitoring data
- Listing, creating, updating, and deleting domain administrators
- Listing, updating, and deleting administration users
- Retrieving licensing information.

Note: The `SUPCluster` interface also contains methods for managing monitoring profiles in a cluster, and monitoring data store policies and domain log data store policies. These methods are described in *Developer Guide for Unwired Server Management API > Code Samples > Monitoring Unwired Platform Components*.

Start Cluster Management

Starts the management of an Unwired Server cluster.

Syntax

```
public static SUPCluster getSUPCluster(ClusterContext
clusterContext) throws SUPAdminException;
```

Examples

- **Cluster startup –** starts the management of the specified cluster.

```
clusterContext = serverContext.getClusterContext("<cluster
name>");
SUPCluster supCluster =
SUPObjectFactory.getSUPCluster(clusterContext);
```

Usage

When an instance of `SUPCluster` is returned from the `SUPObjectFactory`, call its method.

Unwired Servers Retrieval

Retrieves a list of servers that are members in a cluster.

Syntax

```
Collection<ServerVO> getServers() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Getting member servers –** lists the servers that are members of a cluster:
  ```
  Collection<ServerVO> svos = supCluster.getServers();
  ```

Resume an Unwired Server

Resumes an Unwired Server in a cluster.

Syntax

```
void resume(String name) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Resume a server –** resumes an Unwired Server in a cluster:
  ```
  supCluster.resume("<member server name>");
  ```

Suspend an Unwired Server

Suspends a member server in a cluster.

Syntax

```
void suspend(String name) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Suspend a server –** suspends an Unwired Server in a cluster:
  ```
  supCluster.suspend("<member server name>");
  ```

Retrieval of Domains

Retrieves the domains in a cluster.

Syntax

```
Collection<String> getDomains() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of domains –** retrieves the domains in a cluster.

```
Collection<String> domains = supCluster.getDomains();
```

Creation of Domains

Creates domains in a cluster.

Syntax

```
void createDomain(String name) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Creation of domains –** creates,in the cluster, the domain specified by "<domain name>".

```
supCluster.createDomain("<domain name>");
```

Deletion of Domains

Deletes domains from a cluster.

Syntax

```
void deleteDomains(Collection<String> names) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion of domains –** deletes, from the cluster, the domains in the specified array.

```
supCluster.deleteDomains(Arrays.asList(new String[] {
"<domain name 1>", "<domain name 2>" }));
```

Retrieval of Security Configurations

Retrieves a list of security configurations in a cluster.

Syntax

```
Collection<String> getSecurityConfigurations() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of security configurations –** lists the security configurations in a cluster.

  ```
  Collection<String> securityConfigurations=
  supCluster.getSecurityConfigurations();
  ```

Creation of a Security Configuration

Creates a security configuration in a cluster.

Syntax

```
void createSecurityConfiguration(String name) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Creation of a security configuration –** creates a security configuration of the specified name in the cluster:

  ```
  supCluster.createSecurityConfiguration("<security configuration
  name>");
  ```

Deletion of a Security Configuration

Deletes a security configuration from the cluster.

Syntax

```
void deleteSecurityConfigurations(Collection<String> names) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion of a security configuration –** deletes a security configuration from the cluster.
  ```
  supCluster.deleteSecurityConfigurations(securityConfigurations);
  ```

Retrieval of Domain Administrators

Retrieves a list of domain adminstrators in a cluster.

Syntax

```
Collection<DomainAdministratorVO> getDomainAdministrators() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of domain administrators –** retrieves a list of domain administrators in a cluster:
  ```
  //List domain administrators
  for (DomainAdministratorVO davo :
  supCluster.getDomainAdministrators()) {
      System.out.println(davo.getLoginName());
  }
  ```

Creation of a Domain Administrator

Creates a domain adminstrator in the cluster.

Syntax

```
void createDomainAdministrator(DomainAdministratorVO
domainAdministrator) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Creation of a domain administrator –** creates a domain adminstrator in the cluster:

```
//Create a domain administrator
DomainAdministratorVO davo = new DomainAdministratorVO();
davo.setLoginName("<new domain administrator login name>");
supCluster.createDomainAdministrator(davo);
```

Update of a Domain Administrator

Updates a domain administrator in the cluster.

Syntax

```
void updateDomainAdministrator(DomainAdministratorVO
domainAdministrator) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update of a domain adminstrator –** updates a domain administrator in the cluster by setting the login name and company name:

```
//Update a domain administrator
davo = new DomainAdministratorVO();
davo.setLoginName("<domain administrator login name>");
davo.setCompanyName("Sybase");
supCluster.updateDomainAdministrator(davo);
```

Deletion of a Domain Administrator

Deletes a domain administrator from the cluster.

Syntax

```
void deleteDomainAdministrator(DomainAdministratorVO
domainAdministrator) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion of a domain administrator –** deletes the specified domain administrator from the cluster:

```
//Delete a domain administrator
davo = new DomainAdministratorVO();
```

```
davo.setLoginName("<domain administrator login name>");
supCluster.deleteDomainAdministrator(davo);
```

Retrieval and Setting of Authentication Cache Timeout

Retrieves and sets the authentication cache timeout from a cluster.

Syntax

```
Long timeout getAuthenticationCacheTimeout () throws
SUPAdminException;

void setAuthenticationCacheTimeout(user, timeout);
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieve and set authentication cache timeout –** retrieves and sets the specified authentication cache timeout from a cluster:

  ```
  Long timeout = supCluster.getAuthenticationCacheTimeout("admin");
  supCluster.setAuthenticationCacheTimeout("admin", 200L);
  timeout = supCluster.getAuthenticationCacheTimeout("admin");
  assertEquals(new Long(200), timeout);
  ```

Retrieval and Setting of Cluster Properties

Retrieves and sets the properties of a cluster.

Syntax

```
ClusterPropertiesVO getClusterProperties() throws SUPAdminException;

void setClusterSyncDataSharedPathEnabled(boolean) throws
SUPAdminException;

void setClusterSyncDataSharedPath(path) throws SUPAdminException

void setClusterProperties(vo)
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieve or set cluster properties –**

```
//Get cluster properties
ClusterPropertiesVO vo = supCluster.getClusterProperties();
//change cluster properties
vo.setClusterSyncDataSharedPathEnabled(true);
vo.setClusterSyncDataSharedPath("\\\\myhost\\newSharedPath");
//Set cluster properties
supCluster.setClusterProperties(vo);
```

Retrieval and Setting of Maximum Allowed Authentication Failures

Retrieves and sets the maximum number of allowed authentication failures.

Syntax

```
Integer getMaximumAllowedAuthenticationFailure(String
securityConfiguration) throws SUPAdminException;

void setMaximumAllowedAuthenticationFailure(String
securityConfiguration, Integer maximumAllowed) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieve or set cluster properties –**

```
//Get maximum allowed authentication failures
Integer threshold=
supCluster.getMaximumAllowedAuthenticationFailure("admin");
//Set maximum allowed authentication failures
supCluster.setMaximumAllowedAuthenticationFailure("admin", 20);
```

Retrieval and Setting of Authentication Lock Duration

Retrieves and sets the duration for authentication lock.

Syntax

```
Integer getAuthenticationLockDuration(String securityConfiguration)
throws SUPAdminException;

void setAuthenticationLockDuration(String securityConfiguration,
Integer duration) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieve or set authentication lock duration –**

```
Integer duration =
supCluster.getAuthenticationLockDuration("admin");
supCluster.setAuthenticationLockDuration("admin", 3000);
```

Retrieval of Relay Servers

Retrieves a list of Relay Servers configured for an Unwired Server cluster.

Syntax

```
List<RelayServerVO> getRelayServers() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of Relay Servers –** retrieves a list of relay servers in a cluster:

```
// Get all relay servers configured for the Unwired Server
cluster.
List<RelayServerVO> relayServers = supCluster.getRelayServers();
for (RelayServerVO relayServer : relayServers) {
    // Print relay server info
    System.out.println("=========Begin Relay Server
Info=========");
    System.out.println("Host: " + relayServer.getHost());
    System.out.println("HTTP port: " + relayServer.getPort());
    System.out.println("HTTPS port: " +
relayServer.getSecurePort());
    System.out.println("URL suffix: " +
relayServer.getUrlSuffix());
    // Print farm info of this relay server
    System.out.println("======Farms within this relays
server======");
    for (FarmVO farm : relayServer.getFarms()) {
        System.out.println("  " + farm);
        // print server node info of this farm
       System.out.println("===Server nodes within this farm===");
        for (ServerNodeVO serverNode : farm.getServerNodes()) {
            System.out.println("    Server node: " + serverNode);
            // print Outbound Enabler info of this server node
            System.out.println("    Outbound enabler: "
```

```
                    + serverNode.getOutboundEnabler());
        }
    }
    System.out.println("=========End Relay Server Info=========");
}
```

Licensing Information Retrieval

Retrieves information about software and device licensing on Unwired Server.

Syntax

```
LicensingInfoVO getLicensingInfo() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves licensing information for the Unwired Server.

```
// Get Licensing info.
LicensingInfoVO infoVO = supCluster.getLicensingInfo();
System.out.println(infoVO.getAvailableDeviceLicenseCount());
System.out.println(infoVO.getLicenseType());
System.out.println(infoVO.getProductionEdition());
System.out.println(infoVO.getUsedDeviceLicenseCount());
System.out.println(infoVO.getDevicelicenseExpireDate());
System.out.println(infoVO.getServerlicenseExpireDate());
```

Note: For more information on Sybase Unwired Platform licensing, see *System Administration for Sybase Unwired Platform > Systems Maintenance and Monitoring > Platform Licenses*.

Managing Domains

You can manage domains of Unwired Servers through the `SUPDomain` interface. Operations you can perform with this interface include:

- Enabling or disabling a Sybase Unwired Platform domain.
- Packages: listing, creating, deleting, importing, exporting packages.
- Endpoints: listing, creating, deleting, updating endpoints.
- Security configuration: getting/setting associated security configurations.
- Domain administrators: listing administrators.
- Data maintenance: cleaning up accumulated data artifacts.
- Applications: viewing applications and application connections at the domain level.

Start Domain Management

Starts the management of a domain.

Syntax

```
public static SUPDomain getSUPDomain(DomainContext domainContext)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start domain management –** starts the management of the specified domain:

  ```
  DomainContext domainContext =
  serverContext.getDomainContext("<domain name>");
  SUPDomain supDomain =
  SUPObjectFactory.getSUPDomain(domainContext);
  ```

Usage

To manage Unwired Server domains, you must first create an instance of `SUPDomain`.

To perform SUP domain administration operations, you must be assigned an SUP Administrator or SUP Domain Administrator role.

Enable a Domain

Enables a domain.

Syntax

```
void enable(Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Enable a domain –**

  ```
  supDomain.enable(true);  //Enable domain
  ```

Disable a Domain

Disables a domain.

Syntax

```
void enable(Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Disable a domain –**

```
supDomain.enable(false); //Disable domain
```

Package Retrieval

Retrieves a list of packages in a domain.

Syntax

```
Collection<String> getPackages() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Package retrieval –** retrieves a list of packages in a domain:

```
for(String packageName : supDomain.getPackages()){
    System.out.println(packageName);
}
```

Package Deployment

Deploys a package to a domain.

Syntax

```
void deployPackage(String fileName, DEPLOY_MODE deployMode, String
securityConfiguration, Collection<RoleMappingVO> roleMappings,
Map<String, String> endpointMappings) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Package deployment –** deploys a package to a domain:

```
Collection<RoleMappingVO> roleMappingVOs = new
ArrayList<RoleMappingVO>();
RoleMappingVO rmvo1 = new RoleMappingVO();
rmvo1.setSourceRole("Role1");
rmvo1.setRoleMappingType(ROLE_MAPPING_TYPE.AUTO);
RoleMappingVO rmvo2 = new RoleMappingVO();
rmvo2.setSourceRole("Role2");
rmvo2.setRoleMappingType(ROLE_MAPPING_TYPE.AUTO);
RoleMappingVO rmvo3 = new RoleMappingVO();
rmvo3.setSourceRole("Role3");
rmvo3.setRoleMappingType(ROLE_MAPPING_TYPE.AUTO);

roleMappingVOs.add(rmvo1);
roleMappingVOs.add(rmvo2);
roleMappingVOs.add(rmvo3);

Map<String, String> endpointMappings = new HashMap<String,
String>();
endpointMappings.put("sampledb", "sampledb2");

supDomain.deployPackage("<deployment unit file name>",
        DEPLOY_MODE.UPDATE,
        "<security configuration name>", roleMappingVOs,
endpointMappings);
```

Package Deletion

Deletes a package from a domain.

Syntax

```
void deletePackage(String name) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Package deletion –** deletes the specified package from the domain:

```
supDomain.deletePackage("<package name>");
```

Package Import

Imports a package to a domain.

Syntax

```
void importPackage(String fileName, Boolean overwrite) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Package import –** imports a package with the specified package file name to the domain:
  ```
  supDomain.importPackage("<exported package file name>", true);
  ```

Usage

You can only import package into the same domain as the one you exported from. The API requires that the domain where the package was exported from exists on the server when the import is done. Also, you are required to create domains in the same order in both the export and import server environments, which ensures that an internal ID assigned to the domain in both environment matches.

You can verify the internal ID assigned to a domain by looking at the prefix used in the package folder in the zip.

Package Export

Exports a package from a domain.

Syntax

```
void exportPackage(String fileName, String name,
EnumSet<PACKAGE_EXPORT_OPTION> exportOptions) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Package Export –** exports a package with the specified file name, package name, and options from a domain:
  ```
  EnumSet<PACKAGE_EXPORT_OPTION> options =
  EnumSet.noneOf(PACKAGE_EXPORT_OPTION.class);
  options.add(PACKAGE_EXPORT_OPTION.LOG_LEVEL);
  options.add(PACKAGE_EXPORT_OPTION.ROLE_MAPPING);
  options.add(PACKAGE_EXPORT_OPTION.REPLICATION_SUBSCRIPTION_TEMPLA
  TE);
  options.add(PACKAGE_EXPORT_OPTION.PACKAGE_LOGGING );

  supDomain.exportPackage("<file name>", "<package name>",
  options);
  ```

Endpoint Retrieval

Retrieves a list of server connection endpoints in the domain. The supported endpoint types are JDBC, SAP®, and WEBSERVICE.

Syntax

```
Collection<EndpointVO> getEndpoints(ENDPOINT_TYPE type) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint retrieval –** retrieves a list of endpoints for each endpoint type:

```
for(EndpointVO evo : supDomain.getEndpoints(ENDPOINT_TYPE.JDBC)){
    System.out.println(evo.getName());
    System.out.println(evo.getExtraProps());
}

for(EndpointVO evo : supDomain.getEndpoints(ENDPOINT_TYPE.SAP)){
    System.out.println(evo.getName());
    System.out.println(evo.getExtraProps());
}
for(EndpointVO evo :
supDomain.getEndpoints(ENDPOINT_TYPE.WEBSERVICE)){
    System.out.println(evo.getName());
    System.out.println(evo.getExtraProps());
}
```

Note: For detailed information on each of these endpoint types, see *Developer Guide for Unwired Server Management API > Property Reference > EIS Data Source Connection Properties Reference*.

Endpoint Creation

Creates a server connection endpoint of the specified endpoint type.

Syntax

```
void createEndpoint(ENDPOINT_TYPE type, String name, String
template, Map<String, String> properties) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint creation –** creates an endpoint for each endpoint type, and sets its properties:

```
Map<String, String> properties = new HashMap<String, String>();

// For Sybase ASA
properties.put("commitProtocol", "<commit protocol>");
properties.put("dataSourceClass", "<data source class>");
properties.put("databaseURL", "<database URL>");
properties.put("user", "<user name>");
properties.put("password", "<password>");
supDomain.createEndpoint(ENDPOINT_TYPE.JDBC, "<endpoint name>",
"<template name>", properties);

properties.clear();
properties.put("jco.client.user", "<jco client user>");
properties.put("jco.client.passwd", "<jco client password>");
properties.put("jco.client.ashost", "<jco client AS host>");
properties.put("jco.client.client", "<jco client>");
supDomain.createEndpoint(ENDPOINT_TYPE.SAP, "<endpoint name>",
"<template name>", properties);

properties.clear();
properties.put("address", "<address>");
properties.put("user", "<user name>");
properties.put("password", "<password>");
supDomain.createEndpoint(ENDPOINT_TYPE.WEBSERVICE, "<endpoint
name>", "<template name>", properties);
```

Endpoint Deletion

Deletes a specific server connection endpoint of the specified type.

Syntax

```
void deleteEndpoint(ENDPOINT_TYPE type, String name) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint deletion –** deletes an endpoint of each endpoint type:

```
supDomain.deleteEndpoint(ENDPOINT_TYPE.JDBC, "<endpoint name>");
supDomain.deleteEndpoint(ENDPOINT_TYPE.SAP, "<endpoint name>");
supDomain.deleteEndpoint(ENDPOINT_TYPE.WEBSERVICE, "<endpoint
name>");
```

Endpoint Update

Updates the properties of a specific server connection endpoint.

Syntax

```
void updateEndpoint(EndpointVO endpoint) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint update –**

```
EndpointVO evo = new EndpointVO();
evo.setName("sampledb2");
evo.setType(ENDPOINT_TYPE.JDBC);
Map<String, String> properties = new HashMap<String, String>();
properties.put("commitProtocol", "pessimistic");
properties.put("dataSourceClass",
"com.sybase.jdbc3.jdbc.SybDataSource");
properties.put("databaseURL", "jdbc:sybase:Tds:localhost:5500/
sampledb2?ServiceName=sampledb2");
evo.setExtraProps(properties);
supDomain.updateEndpoint(evo);
```

Endpoint Template Retrieval

Retrieves a list of endpoint templates in the domain. The supported endpoint template types are JDBC, SAP®, and WEBSERVICE.

Syntax

```
Collection<EndpointVO> getEndpointTemplates(ENDPOINT_TYPE type)
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint template retrieval –** retrieves a list of endpoint templates for each endpoint type:

```
for (EndpointVO evo : supDomain
        .getEndpointTemplates(ENDPOINT_TYPE.JDBC)) {
    System.out.println(evo.getName());
    System.out.println(evo.getExtraProps());
```

```
}

for (EndpointVO evo :
supDomain.getEndpointTemplates(ENDPOINT_TYPE.SAP)) {
    System.out.println(evo.getName());
    System.out.println(evo.getExtraProps());
}
for (EndpointVO evo : supDomain
        .getEndpointTemplates(ENDPOINT_TYPE.WEBSERVICE)) {
    System.out.println(evo.getName());
    System.out.println(evo.getExtraProps());
}
```

Note: For detailed information on each of these endpoint types, see *Developer Guide for Unwired Server Management API > Property Reference > EIS Data Source Connection Properties Reference*.

Endpoint Template Creation

Creates a server connection endpoint template for the specified endpoint type.

Syntax

```
void createEndpointTemplate(ENDPOINT_TYPE type, String name, String
template, Map<String, String> properties) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint creation –** creates an endpoint for each endpoint type, and sets its properties:

```
Map<String, String> properties = new HashMap<String, String>();
properties.put("commitProtocol", "<commit protocol>");
properties.put("dataSourceClass", "<data source class>");
properties.put("databaseURL", "<database URL>");
properties.put("user", "<user name>");
properties.put("password", "<password>");
supDomain.createEndpointTemplate(ENDPOINT_TYPE.JDBC,
"myJDBC_template",
                "Sybase_ASA_template", properties);

properties.clear();
properties.put("jco.client.user", "<jco client user>");
properties.put("jco.client.passwd", "<jco client password>");
properties.put("jco.client.ashost", "<jco client AS host>");
properties.put("jco.client.client", "<jco client>");
supDomain.createEndpointTemplate(ENDPOINT_TYPE.SAP,
"mySAP_template",
       "sap_template", properties);

properties.clear();
```

```
properties.put("address", "<address>");
properties.put("user", "<user name>");
properties.put("password", "<password>");
supDomain.createEndpointTemplate(ENDPOINT_TYPE.WEBSERVICE,
        "myWS_template", "webservice_template", properties);
```

Endpoint Template Deletion

Deletes a specific server connection endpoint template of the specified type.

Syntax

```
void deleteEndpointTemplate(ENDPOINT_TYPE type, String name) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint template deletion –** deletes an endpoint template of each endpoint type:

```
supDomain.deleteEndpointTemplate(ENDPOINT_TYPE.JDBC,
        "<endpoint template name>");
supDomain.deleteEndpointTemplate(ENDPOINT_TYPE.SAP,
        "<endpoint template name>");
supDomain.deleteEndpointTemplate(ENDPOINT_TYPE.WEBSERVICE,
        "<endpoint template name>");
```

Endpoint Template Update

Updates the properties of a specific server connection endpoint template.

Syntax

```
void updateEndpointTemplate(EndpointVO endpoint) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint update –**

```
EndpointVO evo = new EndpointVO();
evo.setName("<endpoint template name>");
evo.setType(ENDPOINT_TYPE.JDBC);
Map<String, String> properties = new HashMap<String, String>();
properties.put("commitProtocol", "pessimistic");
properties.put("dataSourceClass",
"com.sybase.jdbc3.jdbc.SybDataSource");
```

```
properties.put("databaseURL", "jdbc:sybase:Tds:localhost:5500/
sampledb2?ServiceName=sampledb2");
evo.setExtraProps(properties);
supDomain.updateEndpointTemplate(evo);
```

Retrieval of Security Configurations

Retrieves a list of security configurations for a domain.

Syntax

```
Collection<String> getSecurityConfigurations() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of security configurations –** retrieves a list of security configurations for a domain:

```
for (String securityConfiguration : supDomain
        .getSecurityConfigurations()) {
    System.out.println(securityConfiguration);
}
```

Update of Security Configurations

Updates security configurations in the domain. You must be assigned an SUP Administrator role to perform this operation.

Syntax

```
void setSecurityConfigurations(Collection<String> names) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update of security configurations –** updates the security configurations specified in an array:

```
supDomain.setSecurityConfigurations(Arrays.asList(new String[] {
        "<security configuration 1>", "<security configuration
2>" }));
```

Retrieve Scheduled Purge Task Status

Checks to see whether domain-level cleanup is scheduled for the specified purge task type.

Syntax

```
Boolean isScheduledPurgeTaskEnable(SCHEDULE_PURGE_TASK task) throws
SUPAdminException;
```

Returns

If successful, returns true or false. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge task status –** retrieves the scheduled data purge task status for synchronization cache, subscription, client log, and error history purge tasks.

```
supDomain.isScheduledPurgeTaskEnable(SCHEDULE_PURGE_TASK.CLIENT_L
OG);
supDomain.isScheduledPurgeTaskEnable(SCHEDULE_PURGE_TASK.ERROR_HI
STORY);
supDomain.isScheduledPurgeTaskEnable(SCHEDULE_PURGE_TASK.SUBSCRIP
TION);
supDomain.isScheduledPurgeTaskEnable(SCHEDULE_PURGE_TASK.SYNC_CAC
HE_GROUP);
```

Enable or Disable Scheduled Purge Tasks

Enables or disables domain-level cleanup using the current scheduled purge task values.

Syntax

```
void enableScheduledPurgeTask(SCHEDULE_PURGE_TASK task, Boolean
enabled) throws SUPAdminException;
```

Returns

If successful, enables or disables cleanup. If unsuccessful, returns `SUPAdminException`.

Examples

- **Enables or disables purge tasks –** enables or disables the scheduled data purge tasks for synchronization cache, subscription, client log, or error history.

```
supDomain.enableScheduledPurgeTask(SCHEDULE_PURGE_TASK.CLIENT_LOG
, true);
supDomain.enableScheduledPurgeTask(SCHEDULE_PURGE_TASK.ERROR_HIST
ORY, false);
supDomain.enableScheduledPurgeTask(SCHEDULE_PURGE_TASK.SUBSCRIPTI
ON, false);
```

```
supDomain.enableScheduledPurgeTask(SCHEDULE_PURGE_TASK.SYNC_CACHE
_GROUP, true);
```

Get Purge Task Schedule

Gets the cleanup schedule for the selected purge task type. Getting the purge task schedule is typically used with setting the purge task schedule.

Syntax

```
ScheduleVO getPurgeTaskSchedule(SCHEDULE_PURGE_TASK task) throws
SUPAdminException;
```

Returns

If successful, returns true or false. If unsuccessful, returns `SUPAdminException`.

Examples

- **Get purge task schedule –** gets and sets the purge task schedule for synchronization cache, subscription, client log, or error history.

```
ScheduleVO reschedule =
supDomain.getPurgeTaskSchedule(SCHEDULE_PURGE_TASK.CLIENT_LOG);
reschedule =
supDomain.getPurgeTaskSchedule(SCHEDULE_PURGE_TASK.ERROR_HISTORY)
;
reschedule =
supDomain.getPurgeTaskSchedule(SCHEDULE_PURGE_TASK.SUBSCRIPTION);
reschedule =
supDomain.getPurgeTaskSchedule(SCHEDULE_PURGE_TASK.SYNC_CACHE_GRO
UP);
```

Set Purge Task Schedule

Sets the domain-level cleanup schedule for the selected purge task. Setting the purge task schedule is typically used with getting the purge task schedule.

Syntax

```
void setPurgeTaskSchedule(SCHEDULE_PURGE_TASK task, ScheduleVO
schedule) throws SUPAdminException;
```

Returns

If successful, returns the schedule for the selected type. If unsuccessful, returns `SUPAdminException`.

Examples

- **Set purge task schedule –** gets and sets the purge task schedule for synchronization cache, subscription, client log, or error history.

```
ScheduleVO schedule = new ScheduleVO();
schedule.setDaysofweek(EnumSet.of(DAY_OF_WEEK.MONDAY,DAY_OF_WEEK.
FRIDAY));
schedule.setStartDate(new Date());
schedule.setStartTime(new Date());
schedule.setEndDate(new Date());
schedule.setEndTime(new Date());
schedule.setFreq(SCHEDULE_FREQ.INTERVAL);
schedule.setInterval(50);

supDomain.setPurgeTaskSchedule(SCHEDULE_PURGE_TASK.CLIENT_LOG,
schedule);
supDomain.setPurgeTaskSchedule(SCHEDULE_PURGE_TASK.ERROR_HISTORY,
schedule);
supDomain.setPurgeTaskSchedule(SCHEDULE_PURGE_TASK.SUBSCRIPTION,
schedule);
supDomain.setPurgeTaskSchedule(SCHEDULE_PURGE_TASK.SYNC_CACHE_GRO
UP, schedule);
```

Purge Synchronization Cache

Purges synchronization cache at the domain level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeSyncCacheGroup(Boolean synchronous) throws
SUPAdminException;
```

Returns

If successful, purges synchronization cache using the schedule. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge sync cache –** purges the synchronization cache using defined settings.

```
supDomain.purgeSyncCacheGroup(false);
```

Purge Client Log

Purges the client log at the domain level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeClientLog(ClientLogPurgeOptionVO purgeOption, Boolean
synchronous) throws SUPAdminException;
```

Returns

If successful, purges the client log using the schedule. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge client log –** purges the client log using current settings.

```
ClientLogPurgeOptionVO purgeOption = new
ClientLogPurgeOptionVO();
purgeOption.setDaysToPreserve(10);
supDomain.purgeClientLog(purgeOption, false);
```

Get Client Log Purge Options

Obtains the current client log purge settings at the domain level.

Syntax

```
ClientLogPurgeOptionVO getClientLogPurgeOption() throws
SUPAdminException;
```

Returns

If successful, gets the current client log purge settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Gets client log options –** gets the current client log purge options.

```

ClientLogPurgeOptionVO roption =
supDomain.getClientLogPurgeOption();
```

Set Client Log Purge Options

Sets the client log purge options at the domain level using the current settings.

Syntax

```
void setClientLogPurgeOption(ClientLogPurgeOptionVO option) throws
SUPAdminException;
```

Returns

If successful, sets the current client log purge settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Sets client log options –** sets the current client log purge settings, which includes preserving data for the last 15 days.

```
ClientLogPurgeOptionVO option = new ClientLogPurgeOptionVO();
option.setDaysToPreserve(15);
supDomain.setClientLogPurgeOption(option);
```

Purge Error History

Purges the error history at the domain level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeErrorHistory(ErrorHistoryPurgeOptionVO purgeOption,
Boolean synchronous) throws SUPAdminException;
```

Returns

If successful, purges the error history using the schedule. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge error history –** purges the error history using defined settings.

```
ErrorHistoryPurgeOptionVO purgeOption = new
ErrorHistoryPurgeOptionVO();
purgeOption.setDaysToPreserve(10);
supDomain.purgeErrorHistory(purgeOption, false);
```

Get Error History Purge Options

Gets the current error history purge option settings at the domain level.

Syntax

```
ErrorHistoryPurgeOptionVO getErrorHistoryPurgeOption() throws
SUPAdminException;
```

Returns

If successful, gets the current error history purge settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Gets error history purge options –** gets the current error history purge settings.

```
ErrorHistoryPurgeOptionVO roption =
supDomain.getErrorHistoryPurgeOption();
```

Set Error History Purge Options

Sets the error history purge options at the domain level using current settings.

Syntax

```
void setErrorHistoryPurgeOption(ErrorHistoryPurgeOptionVO option)
throws SUPAdminException;
```

Returns

If successful, sets the current error history purge settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Set error history purge options –** sets the current error history purge settings.

  ```
  ErrorHistoryPurgeOptionVO option = new
  ErrorHistoryPurgeOptionVO();
  option.setDaysToPreserve(15);
  supDomain.setErrorHistoryPurgeOption(option);
  ```

Purge Subscription

Purges subscriptions at the domain level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeSubscription(SubscriptionPurgeOptionVO purgeOption,
Boolean synchronous) throws SUPAdminException;
```

Returns

If successful, purges subscriptions using the schedule. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge subscription –** purges subscriptions using defined settings.

  ```
  SubscriptionPurgeOptionVO purgeOption = new
  SubscriptionPurgeOptionVO();
  purgeOption.setDaysInactive(10);
  supDomain.purgeSubscription(purgeOption, false);
  ```

Get Subscription Purge Options

Obtains the current subscription purge options at the domain level.

Syntax

```
SubscriptionPurgeOptionVO getSubscriptionPurgeOption() throws
SUPAdminException;
```

Returns

If successful, gets the subscription purge settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Gets subscription purge options –** gets the current subscription purge settings.

  ```
  SubscriptionPurgeOptionVO roption =
  supDomain.getSubscriptionPurgeOption();
  ```

Set Subscription Purge Options

Sets the subscription purge options at the domain level.

Syntax

```
void setSubscriptionPurgeOption(SubscriptionPurgeOptionVO option)
throws SUPAdminException;
```

Returns

If successful, sets the current subscription purge settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Sets subscription purge options –** sets the subscription purge options, including setting 15 as the number of inactive days.

  ```
  SubscriptionPurgeOptionVO option = new
  SubscriptionPurgeOptionVO();
  option.setDaysInactive(15);
  supDomain.setSubscriptionPurgeOption(option);
  ```

Managing Packages

You can manage MBO packages and their properties through the `SUPPackage` interface. Operations you can perform with this interface include:

- **Security configuration –** getting or setting security configuration.
- **Synchronization group –** getting or setting synchronization group properties.
- **Synchronization tracing –** enabling or disabling synchronization tracing.
- **Message-based sync subscription management –** these subscriptions determine what synchronization messages mobile device users receive on messaging-based devices.
- **Replication-based sync subscription and template management –** these subscriptions determine what synchronization messages mobile device users receive on replication-based devices.
- **Package role mapping –** getting/setting package level role mappings. You can define role mapping for the package to map logical roles in the package to physical roles on the Unwired Server.
- **Applications –** viewing applications, adding or removing application to/from a package, viewing application users.
- **Uncategorized –** enabling and disabling packages, listing MBOs, managing cache groups, listing personalization keys, and retrieving endpoint properties.

Start Package Management

Starts the management of an Unwired Server package.

Syntax

```
public static SUPPackage getSUPPackage(PackageContext
packageContext) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start package management –**

```
domainContext = serverContext.getDomainContext("<domain name>");
packageContext = domainContext.getPackageContext("<package
name>");
SUPPackage suppkg =
SUPObjectFactory.getSUPPackage(packageContext);
```

Usage

To manage Unwired Server packages, you must first create an instance of `SUPPackage`.

Enable a Package

Enables a package.

Syntax

```
void enable(Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Enable a package –** enables a package and retrieves a list of mobile business objects and personalization keys in the package.

```
//Enable a package.
suppkg.enable(true); //Enable package

//Retrieve a list of MBOs
for (String mboName : suppkg.getMobileBusinessObjects()) {
    System.out.println(mboName);
}
//Retrieve a list of personalization keys
for(PersonalizationKeyVO pvo : suppkg.getPersonalizationKeys()){
    System.out.println(pvo.getKey());
```

Disable a Package

Disables a package.

Syntax

```
void enable(Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Disable a package –**

```
//Disable a package.
suppkg.enable(false); //Disable package
```

Enable Synchronization Tracing

Enables synchronization tracing.

Syntax

```
void setSyncTracingStatus(Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Enable synchronization tracing –**

  ```
  suppkg.setSyncTracingStatus(true); //Enable synchronization
  tracing
  ```

Disable Synchronization Tracing

Disables synchronization tracing.

Syntax

```
void setSyncTracingStatus(Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Disable synchronization tracing –**

  ```
  suppkg.setSyncTracingStatus(false); //Disable synchronization
  tracing
  ```

Retrieval of Security Configurations

Retrieves a list of security configurations for a package.

Syntax

```
String getSecurityConfiguration() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of security configurations –**
  ```
  String securityConfiguration = suppkg.getSecurityConfiguration();
  ```

Set Security Configuration

Sets the security configuration for a package.

Syntax

```
void setSecurityConfiguration(String name) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Set security configuration –**
  ```
  suppkg.setSecurityConfiguration("<security configuration name>");
  ```

Retrieval of Synchronization Group Properties

Retrieves a list of synchronization group properties for a package.

Syntax

```
Collection<SyncGroupVO> getSyncGroups() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of synchronization group properties –**
  ```
  for(SyncGroupVO sgvo : suppkg.getSyncGroups()){
      System.out.println(sgvo.getName());
  }
  ```

Set Synchronization Group Properties

Sets properties for a synchronization group in a package.

Syntax

```
void setSyncGroupChangeDetectionInterval(String syncGroup, Integer
checkInterval) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Set synchronization group properties –** updates the check interval for the specified synchronization group:

```
suppkg.setSyncGroupChangeDetectionInterval("<sync group name>",
1000);
```

Retrieval of Messaging Package Subscriptions

Retrieves messaging package subscriptions.

Syntax

```
Collection<MBSSubscriptionVO> getMBSSubscriptions() throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of messaging package subscriptions –**

```
Collection<MBSSubscriptionVO> mbsSubs =
suppkg.getMBSSubscriptions();
MBSSubscriptionVO mbsSub = suppkg.getMBSSubscription("<client
id>");
```

Note: For more information on managing messaging package subscriptions, see *Sybase Unwired Platform Systems Administration Guide > System Administration > Package Administration > Managing Deployed Package Subscriptions*.

Deletion of Messaging Package Subscriptions

Deletes messaging package subscriptions.

Syntax

```
void removeMBSSubscriptions(Collection<String> clientIds) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion of messaging package subscriptions –**
  ```
  suppkg.removeMBSSubscriptions(clientIds);
  ```

Suspend Package Subscriptions

Suspends messaging package subscriptions, or DOE-C package subscriptions.

Syntax

```
void suspendMBSSubscriptions(Collection<String> clientIds) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Suspend messaging (or DOE-C) package subscriptions –**
  ```
  suppkg.suspendMBSSubscriptions(clientIds);
  ```

Resume Package Subscriptions

Resumes messaging package subscriptions, or DOE-C package subscriptions.

Syntax

```
void resumeMBSSubscriptions(Collection<String> clientIds) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Resume messaging (or DOE-C) package subscriptions –**
  ```
  suppkg.resumeMBSSubscriptions(clientIds);
  ```

Reset Messaging Package Subscriptions

Resets messaging package subscriptions.

Syntax

```
void resetMBSSubscriptions(Collection<String> clientIds) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Reset messaging package subscriptions –**

```
suppkg.resetMBSSubscriptions(clientIds);
```

Retrieval of Replication Package Subscriptions

Retrieves replication package subscriptions.

Syntax

```
Collection<RBSSubscriptionVO> getRBSSubscriptions(String syncGroup,
String user) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of replication package subscriptions –**

```
for (RBSSubscriptionVO rbsSub : suppkg
        .getRBSSubscriptions("<sync group name>")) {
    System.out.println(rbsSub.getSyncGroup() + ":"
            + rbsSub.getClientId());
}
for (RBSSubscriptionVO rbsSub : suppkg.getRBSSubscriptionVOs(
        "<sync group name>", "<user name>")) {
    System.out.println(rbsSub.getSyncGroup() + ":"
            + rbsSub.getClientId());
}
```

Note: For more information on managing messaging package subscriptions, see *Sybase Unwired Platform Systems Administration Guide > System Administration > Package Administration > Managing Deployed Package Subscriptions*.

Update of Replication Package Subscriptions

Updates replication package subscriptions.

Syntax

```
void updateRBSSubscription(RBSSubscriptionVO rbsSub) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update of replication package subscriptions –** updates subscriptions of replication packages and sets the properties:

```
RBSSubscriptionVO rbsSub = new RBSSubscriptionVO();
//Client id, sync group, package and domain can uniquely
//identify a RBS subscription
rbsSub.setClientId("<client id>");
rbsSub.setSyncGroup("<sync group>");
//Bellow are the modifiable properties of a RBS subscription
//Please refer to Java doc for detailed information.
rbsSub.setAdminLocked(false);
rbsSub.setPushEnabled(true);
rbsSub.setSyncIntervalMinutes(5);
suppkg.updateRBSSubscription(rbsSub);
```

Removal of Replication Package Subscriptions

Removes a subscription or a list of subscriptions for a package.

Syntax

```
void removeRBSSubscription(String syncGroup, String clientId) throws
SUPAdminException;

void removeRBSSubscriptions(String syncGroup) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Removal of replication package subscriptions –** shows how to remove a list of subscriptions, or a single subscription, for a replication package:

```
//Remove one subscription
suppkg.removeRBSSubscription("<sync group name>", "<client id>"

//Remove a list of subscriptions
suppkg.removeRBSSubscriptions(Arrays.asList(new String[] {
        "<client id 1>", "<client id 2>" }));
suppkg.removeRBSSubscriptions("<sync group>");
suppkg.removeRBSSubscriptions("<sync group>", "<user name>");
```

Purge RBS and MBS Subscriptions

Purges replication-based and message-based synchronization (RBS and MBS) subscriptions at the package level using the number of inactive days. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeSubscription(SubscriptionPurgeOptionVO purgeOption,
Boolean synchronous) throws SUPAdminException;
```

Returns

If successful, purges RBS and MBS subscriptions based on the number of inactive days specified. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge subscriptions –** purges RBS and MBS subscriptions.

  ```
  SubscriptionPurgeOptionVO purgeOption = new
  SubscriptionPurgeOptionVO();
  purgeOption.setDaysInactive(10);
  suppkg.purgeSubscription(purgeOption, false);
  ```

Create Subscription Templates

Creates a subscription template for replication packages.

Syntax

```
RBSSubscriptionVO createRBSSubscriptionTemplate(String syncGroup,
Boolean isPushEnabled, Boolean isAdminLocked, Integer
minimumSyncMinutes) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Creation of a subscription template –** creates a subscription template for replication packages:

  ```
  suppkg.createRBSSubscriptionTemplate("<sync group name>", false,
  false,    5);
  ```

Retrieval of Role Mappings

Retrieves role mappings for a package.

Role mappings map logical roles in the package to physical roles on the Unwired Server.

Syntax

```
Collection<RoleMappingVO> getRoleMappings() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of role mappings –**

```
Collection<RoleMappingVO> roleMappingVOs =
suppkg.getRoleMappings();
```

Note: See the *Sybase Unwired Platform Systems Administration Guide > Security Administration > Security Layers > Roles and Mappings*.

Set Role Mappings

Sets role mappings for a package.

Syntax

```
void setRoleMappings(Collection<RoleMappingVO> rmvos) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Set role mappings –**

```
roleMappingVOs = new ArrayList<RoleMappingVO>();
RoleMappingVO rmvo1 = new RoleMappingVO();
rmvo1.setSourceRole("Role1");
rmvo1.setRoleMappingType(ROLE_MAPPING_TYPE.AUTO);
RoleMappingVO rmvo2 = new RoleMappingVO();
rmvo2.setSourceRole("Role2");
rmvo2.setRoleMappingType(ROLE_MAPPING_TYPE.AUTO);
RoleMappingVO rmvo3 = new RoleMappingVO();
rmvo3.setSourceRole("Role3");
rmvo3.setRoleMappingType(ROLE_MAPPING_TYPE.AUTO);

roleMappingVOs.add(rmvo1);
roleMappingVOs.add(rmvo2);
roleMappingVOs.add(rmvo3);

suppkg.setRoleMappings(roleMappingVOs);
```

Cache Groups

A cache group specifies the data refresh behavior for every mobile business object (MBO) within that group.

You can perform these management tasks for cache groups:

- Retrieving a list of cache groups
- Managing schedule properties of a cache group
- Listing the MBOs associated with a cache group
- Purging or clearing a cache group

Cache Groups Retrieval

Retrieves a list of cache groups for a package.

Syntax

```
Collection<CacheGroupVO> getCacheGroups() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of cache groups –**

```
for(CacheGroupVO cgvo : suppkg.getCacheGroups()){
    System.out.println(cgvo.getName());
}
```

Schedule Properties Retrieval

Retrieves the schedule properties of a cache group for a package.

Syntax

```
CacheGroupScheduleVO getCacheGroupSchedule(String cacheGroupName)
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of schedule properties –** retrieves a list of cache groups for a package:

```
CacheGroupScheduleVO cgsvo = suppkg
        .getCacheGroupSchedule("<cache group name>");
```

Set Schedule Properties

Sets the schedule properties of a cache group for a package.

Syntax

```
void setCacheGroupSchedule(String cacheGroupName,
CacheGroupScheduleVO cacheGroupSchedule) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Set schedule properties –** retrieves a list of cache groups for a package:

```
CacheGroupScheduleVO cgsvo = new CacheGroupScheduleVO();
cgsvo.setFrequency(SCHEDULE_FREQ.DAILY);

EnumSet<DAY_OF_WEEK> daysOfWeek =
EnumSet.noneOf(DAY_OF_WEEK.class);
daysOfWeek.add(DAY_OF_WEEK.MONDAY);
daysOfWeek.add(DAY_OF_WEEK.THURSDAY);
cgsvo.setDayOfWeek(daysOfWeek);

//start date: 2009-12-03
//start time: 18:31:45
//end date: 2009-12-23
//end time: 21:34:47
Calendar cal = Calendar.getInstance();
cal.set(Calendar.YEAR, 2009);
cal.set(Calendar.MONTH, 11);
cal.set(Calendar.DAY_OF_MONTH, 3);
Date startDate = cal.getTime();
cgsvo.setStartDate(startDate);

cal.set(Calendar.YEAR, 2009);
cal.set(Calendar.MONTH, 11);
cal.set(Calendar.DAY_OF_MONTH, 23);
Date endDate = cal.getTime();
cgsvo.setEndDate(endDate);

cal.set(Calendar.HOUR_OF_DAY, 18);
cal.set(Calendar.MINUTE, 31);
cal.set(Calendar.SECOND, 45);
Date startTime = cal.getTime();
cgsvo.setStartTime(startTime);

cal.set(Calendar.HOUR_OF_DAY, 21);
cal.set(Calendar.MINUTE, 34);
cal.set(Calendar.SECOND, 47);
Date endTime = cal.getTime();
```

```
cgsvo.setEndTime(endTime);

suppkg.setCacheGroupSchedule("<cache group name>", cgsvo);
```

- **Set cache group interval –**

```
CacheGroupScheduleVO cgsvo = new CacheGroupScheduleVO();
cgsvo.setFrequency(SCHEDULE_FREQ.INTERVAL);
cgsvo.setInterval(CacheGroupScheduleVO.NEVER_EXPIRE);
suppkg.setCacheGroupSchedule("<cache group name>", cgsvo);
```

Associated Mobile Business Objects

Retrieves a list of the mobile business objects associated with a cache group.

Syntax

```
Collection<String> getCacheGroupMBOs(String cacheGroupName) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Getting associated mobile business objects –**

```
for(String mboName : suppkg.getCacheGroupMBOs("<cache group
name>")){
    System.out.println(mboName);
}
```

Cache Group Purge

Physically deletes rows in the cache group that are marked as logically deleted and are older than the specified date.

Syntax

```
void purgeCacheGroup(String cacheGroupName, Date date) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Cache group purge –** physically deletes data that is marked as deleted and older than the `dateThreshold`:

```
Calendar cal = Calendar.getInstance();
cal.clear();
```

```
cal.set(Calendar.YEAR, 2009);
cal.set(Calendar.MONTH, 11);
cal.set(Calendar.DAY_OF_MONTH, 3);
Date dateThreshold = cal.getTime();
// Physically delete data that is marked as deleted and older than
the
// dateThreshold
suppkg.purgeCacheGroup("<cache group name>", dateThreshold);
```

Usage

Ensure that all devices have synchronized at least once before the specified purge date.

Mobile Business Objects

Packages contain mobile business objects that are deployed to Unwired Server to facilitate access to back-end data and transactions from mobile devices.

Note: See the *Sybase Unwired Platform Systems Administration Guide > System Administration > Package Administration > MBO Package Management Overview.*

Mobile Business Object Retrieval

Retrieves a list of mobile business objects for a package.

Syntax

```
Collection<String> getMobileBusinessObjects() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile business object retrieval –**

```
//Retrieve a list of MBOs
for (String mboName : suppkg.getMobileBusinessObjects()) {
    System.out.println(mboName);
}
```

Personalization Keys

Personalization keys are created by the MBO developer for use as client parameters (user data, such as user name and password), to be validated by the EIS.

Personalization Key Retrieval

Retrieves a list of personalization keys for a package.

Syntax

```
Collection<PersonalizationKeyVO> getPersonalizationKeys() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Personalization key retrieval –**

```
//Retrieve a list of personalization keys
for(PersonalizationKeyVO pvo : suppkg.getPersonalizationKeys()){
    System.out.println(pvo.getKey());
}
```

Client Logs

Client logs record errors, history, and informational messages for mobile clients. Logs include data change notification logs, device notification logs, error logs, messaging logs, replication logs, and subscription logs.

You can perform these management tasks for client logs:

- Retrieving client logs
- Deleting client logs
- Exporting client logs

Retrieval of Client Logs

Retrieves the client logs specified in the search and sort criteria.

Syntax

```
PaginationResult<LogEntryVO>
getClientLogs(ClientLogSearchCriteriaVO searchCriteria, Integer
skip, Integer take, ClientLogSortVO sortInfo) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Client log retrieval –**

```
//Prepare the search and sort criteria
ClientLogSearchCriteriaVO searchCriteria = new
ClientLogSearchCriteriaVO();
searchCriteria.setUserName("*sup*");
searchCriteria.setLevel("*N?O");
searchCriteria.setOperation("*up*");
ClientLogSortVO sortInfo = new ClientLogSortVO();
sortInfo.setAscending(false);
sortInfo.setSortField(ClientLogSortVO.SortField.device);

//Get client Log
PaginationResult<LogEntryVO> result = suppkg.getClientLogs(
searchCriteria, 0, 5, sortInfo);
```

Deletion of Client Logs

Deletes client logs.

Syntax

```
void deleteClientLogs(List<Long> messageIDs) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Client log deletion –**

```
//Delete Client Log
List<Long> messageIDs = new ArrayList<Long>();
messageIDs.add(310004L);
suppkg.deleteClientLogs(messageIDs);

Map<CLIENT_LOG_FIELD, String> map = new HashMap<CLIENT_LOG_FIELD,
String>();
map.put(CLIENT_LOG_FIELD.USER, "supAdmin");
map.put(CLIENT_LOG_FIELD.START_TIME, "2011-07-07");
map.put(CLIENT_LOG_FIELD.END_TIME, "2011-07-08");
suppkg.deleteClientLogs(map);
```

Export of Client Logs

Exports client logs.

Syntax

```
void exportClientLogs(File file, ClientLogSearchCriteriaVO
searchCriteria, Integer skip, Integer take, ClientLogSortVO
sortInfo) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Client log export –**

```
//Export client Log
suppkg.exportClientLogs(new File("F:/tmp/out.txt"),
searchCriteria, 0,
        3, sortInfo);
```

Purge Client Log

Purges the client log at the package level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeClientLog(ClientLogPurgeOptionVO purgeOption, Boolean
synchronous) throws SUPAdminException;
```

Returns

If successful, purges the client log using current settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge client log –** purges the client log, except for data from the last 10 days.

```
ClientLogPurgeOptionVO purgeOption = new
ClientLogPurgeOptionVO();
purgeOption.setDaysToPreserve(10);
suppkg.purgeClientLog(purgeOption, false);
```

Purge Synchronization Cache

Purges synchronization cache at the package level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeSyncCacheGroup(Boolean synchronous) throws
SUPAdminException;
```

Returns

If successful, purges synchronization cache using current settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge sync cache –** purges the synchronization cache using defined settings.

```
suppkg.purgeSyncCacheGroup(false);
```

Purge Error History

Purges the error history at the package level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeErrorHistory(ErrorHistoryPurgeOptionVO purgeOption,
Boolean synchronous) throws SUPAdminException;
```

Returns

If successful, purges the error history using current settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge error history –** purges the error history, except for data from the last 10 days.

```
ErrorHistoryPurgeOptionVO purgeOption = new
ErrorHistoryPurgeOptionVO();
purgeOption.setDaysToPreserve(10);
suppkg.purgeErrorHistory(purgeOption, false);
```

Purge Subscription

Purges subscriptions at the package level. The purge can be done synchronously or asynchronously.

Syntax

```
void purgeSubscription(SubscriptionPurgeOptionVO purgeOption,
Boolean synchronous) throws SUPAdminException;
```

Returns

If successful, purges subscriptions using current settings. If unsuccessful, returns `SUPAdminException`.

Examples

- **Purge subscription –** purges subscriptions, except for data from the last 10 days.

```
SubscriptionPurgeOptionVO purgeOption = new
SubscriptionPurgeOptionVO();
```

```
purgeOption.setDaysInactive(10);
suppkg.purgeSubscription(purgeOption, false);
```

Add Applications to the Package

Adds existing applications to the package.

Syntax

```
void addApplications(Collection<String> appIds) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Addition of applications to the package –**

```
Collection<String> apps = new ArrayList<String>();
apps.add("app1");
suppkg.addApplications(apps);
```

Remove Applications from the Package

Removes existing applications from the package.

Syntax

```
void removeApplications (Collection<String> appIds) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Remove applications from the package –**

```
Collection<String> apps = new ArrayList<String>();
apps.add("app1");
suppkg.removeApplications(apps);
```

Retrieval of a List of Applications

Retrieves a list of applications for a package.

Syntax

```
Collection<ApplicationVO> getApplications() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of role mappings –**
  ```
  Collection<ApplicationVO> apps = suppkg.getApplications();
  ```

Retrieval of a List of Package Users

Retrieves a list of package users for a package.

Syntax

```
PaginationResult<PackageUserVO> getPackageUsers(PackageUser_SortVO
filter, Long offset, Integer length) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of role mappings –**
  ```
  PackageUser_SortVO filter = new PackageUser_SortVO();
  filter.setSortField(PACKAGE_USER.REGISTRATION_TIME);
  filter.setSortOrder(SORT_ORDER.ASCENDING);
  PaginationResult<PackageUserVO> apps =
  suppkg.getPackageUsers(filter, 0L, 100);
  ```

Managing Mobile Business Objects

You can manage mobile business objects and their properties through the `SUPMobileBusinessObject` interface. Operations you can perform with this interface include:

- **Mobile business objects –** retrieving properties and data refresh history, and listing operations.
- **Endpoints –** retrieving properties.

Start Mobile Business Object Management

Starts the management of a mobile business object.

Syntax

```
public static SUPMobileBusinessObject
getSUPMobileBusinessObject(MBOContext mboContext) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start mobile business object management –**

```
domainContext = clusterContext.getDomainContext("<domain name>");
packageContext = domainContext.getPackageContext("<package
name>");
mboContext = packageContext.getMBOContext("<MBO name>");
SUPMobileBusinessObject supmbo =
SUPObjectFactory.getSUPMobileBusinessObject(mboContext);
```

Usage

To manage Unwired Server mobile business objects, you must first create an instance of `SUPMobileBusinessObject`.

Properties Retrieval

Retrieves properties for a mobile business object.

Syntax

```
MobileBusinessObjectVO getProperties() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Properties retrieval –** retrieves properties for a mobile business object, including name, package, creation date, and roles used:

```
MobileBusinessObjectVO mbovo = supmbo.getProperties();
System.out.println(mbovo.getName());
System.out.println(mbovo.getPackage());
System.out.println(mbovo.getCreationDate());
System.out.println(mbovo.getUsedRoles());
```

Endpoints

Endpoint connection information allows applications to retrieve data from back-end production systems.

Note: For more information, see *Sybase Unwired Platform Systems Administration Guide > Environment Setup > EIS Connections > Changing Connections to Production Data Sources*.

Endpoint Properties Retrieval

Retrieves the properties of an endpoint used by a mobile business object.

Syntax

```
EndpointVO getEndpoint() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint properties retrieval –**

```
EndpointVO evo = supmbo.getEndpoint();
System.out.println(evo.getName());
System.out.println(evo.getType());
for(Map.Entry<String, String> entry :
evo.getExtraProps().entrySet()){
    System.out.println(entry.getKey() + " --> " +
entry.getValue());
}
```

Retrieval of Data Refresh Error History

Retrieves the data refresh error history for a mobile business object.

Syntax

```
Collection<DataRefreshErrorVO> getDataRefreshErrors(Date startDate,
Date endDate) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **History retrieval –**

```
for(DataRefreshErrorVO drevo : supmbo.getDataRefreshErrors(null,
null)){
    System.out.println(drevo.getErrorMessage());
}
```

Deletion of Data Refresh Error History

Deletes the data refresh error history for a mobile business object.

Syntax

```
void deleteDataRefreshErrors(Date startDate, Date endDate) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **History deletion –**

```
supmbo.deleteDataRefreshErrors(null, null);
```

Operations Retrieval

Retrieves a list of the operations of a mobile business object.

Syntax

```
Collection<String> getOperations() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Operations retrieval –**

```
for (String op : supmbo.getOperations()) {
    System.out.println(op);
}
```

Managing Operations

You can manage operations and endpoints used by those operations through the `SUPOperation` interface. Operations you can perform with this interface include:

- **Operations –** retrieving properties.
- **Endpoints –** retrieving properties.

Start Operations Management

Starts the management of an Unwired Server operation.

Syntax

```
public static SUPOperation getSUPOperation(OperationContext
operationContext) throws SUPAdminException
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start operation management –**

```
domainContext = serverContext.getDomainContext("<domain name>");
packageContext = domainContext.getPackageContext("<package
name>");
mboContext = packageContext.getMBOContext("<MBO name>");
operationContext = mboContext.getOperationContext("<operation
name>");
SUPOperation supOperation =
SUPObjectFactory.getSUPOperation(operationContext);
```

Usage

To manage Unwired Server operations, you must first create an instance of `SUPOperation`.

Operation Properties Retrieval

Retrieves the properties of an operation.

Syntax

```
OperationVO getProperties() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Operation properties retrieval –**

```
OperationVO ovo = supOperation.getProperties();
```

Endpoint Properties Retrieval

Retrieves the properties of an endpoint used by an operation.

Syntax

```
EndpointVO getEndpoint() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Endpoint properties retrieval –**

```
EndpointVO evo = supOperation.getEndpointVO();

System.out.println(evo.getExtraProps());
```

Retrieval of Playback Error History

Retrieves the playback error history of an operation.

Syntax

```
Collection<PlaybackErrorVO> getPlaybackErrors(Date startDate, Date
endDate) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Playback history retrieval –**

```
for(PlaybackErrorVO pbevo : supOperation.getPlaybackErrors(null,
null)){
```

```
    System.out.println(pbevo.getErrorMessage());
}
```

Managing Applications and Application Connections and Templates

You can manage applications, application connections, and application connection templates through the `SUPApplication` method. Operations you can perform with this interface include:

- **Managing applications –** creating, deleting, and updating applications. Retrieving a list of applications or application users. Deleting application users. Assigning or unassigning domains to an application. Adding or removing packages from an application, or retrieving a list of packages from an application.
- **Managing application connections –** retrieving, cloning, registering, updating, locking, unlocking, and deleting application connections.
- **Managing application connection templates –** managing, listing, and updating application connection templates.

Start Application Management

Starts the management of Unwired Server applications, application connections, and application connection templates.

Syntax

```
public static SUPApplication getSUPApplication(ClusterContext
clusterContext) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start applicatio management –**

  ```
  app = SUPObjectFactory.getSUPApplication(clusterContext);
  ```

Usage

To manage Unwired Server applications, you must first create an instance of `SUPApplication`.

Managing Applications

Use the `SUPApplication` interface to manage applications. Operations you can perform with this interface include:

- Creating an application
- Deleting an application
- Updating an application
- Retrieving a list of applications
- Retrieving a list of application users
- Deleting application users
- Assigning or unassigning domains from an application
- Retrieving domains assigned to an application
- Adding packages to or removing packages from an application
- Retrieving a list of packages from an application

Application Creation

Creates an application.

Syntax

```
void createApplication(String appID, String displayName, String
description) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Create application –**

  ```
  supApplication.createApplication("app1", "app1display", "app1
  description");
  ```

Application Deletion

Deletes applications.

Syntax

```
void deleteApplications(Collection<String> appIDs) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Delete application –**

```
Collection<String> appIDs = new ArrayList<String>();
appIDs.add("app1");

supApplication.deleteApplications(appIDs);
```

Application Update

Updates the application's display name and description.

Syntax

```
void updateApplication(String appId, String displayName, String
description) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supApplication.updateApplication("app1", "updated desc");
```

Retrieval of a List of Applications

Retrieves a list of applications that satisfy the filter. The return result is paginated.

Syntax

```
PaginationResult<ApplicationVO>
getApplications(ApplicationFilterSortVO filter,
Long offset, Integer length) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
ApplicationFilterSortVO filter = new ApplicationFilterSortVO();
FilterExpression<APPLICATION> resultExpression = new
FilterExpression<APPLICATION>();
FilterExpression<APPLICATION> expression1 = new
FilterExpression<APPLICATION>();
FilterExpression<APPLICATION> expression2 = new
FilterExpression<APPLICATION>();
expression1 = expression1.eq(APPLICATION.APPLICATION_USER,
```

```
"WM2@admin");
expression2 = expression2.eq(APPLICATION.APPLICATION_USER,
"abc@admin");
resultExpression = expression1.or(expression2);

filter.setFilterExpression(resultExpression);
filter.setSortField(APPLICATION.APPLICATION_ID);
filter.setSortOrder(SORT_ORDER.ASCENDING);
PaginationResult<ApplicationVO> apps =
supApplication.getApplications(filter, 0l, 100);
```

Retrieval of a List of Application Users

Retrieves a list of application users.

Syntax

```
PaginationResult<ApplicationVO>
getApplicationUsers(ApplicationUser_FilterSortVO filter, Long
offset, Integer length) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
ApplicationUserFilterSortVO filter = new
ApplicationUserFilterSortVO();

filter.setFilterExpression(null);
filter.setSortField(APPLICATION_USER.APPLICATION_ID);
filter.setSortOrder(SORT_ORDER.ASCENDING);
PaginationResult<ApplicationUserVO> apps =
supApplication.getApplicationUsers(filter, 0l,
100);
```

Application Users Deletion

Deletes a list of application users.

Syntax

```
void deleteApplicationUsers(Collection<ApplicationUserVO> users)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

```
Collection<ApplicationUserVO> users = new
ArrayList<ApplicationUserVO>();
ApplicationUserVO user1 = new ApplicationUserVO();
user1.setApplicationId("app1");
user1.setSecurityConfiguration("admin");
user1.setUserName("user1");
users.add(user1);
supApplication.deleteApplicationUsers(users);
```

Assign Domains to an Application

Assigns domains to the specified application.

Syntax

```
void assignDomainsToApplication(String appID, Collection<String>
domains) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Assign Domains –**

```
Collection<String> domains = new ArrayList<String>();
domains.add("default");
domains.add("domain1");
supApplication.assignDomainsToApplication("app1", domains);
```

Unassign Domains from an Application

Unassigns domains from the specified application.

Syntax

```
void unassignDomainsToApplication(String appID, Collection<String>
domains) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Unassign domains –**

```
Collection<String> domains = new ArrayList<String>();
domains.add("default");
```

```
domains.add("domain1");
supApplication.unassignDomainsFromApplication("app1", domains);
```

Retrieval of Assigned Domains

Retrieves the domains assigned to an application.

Syntax

```
Collection<String> getApplicationDomainAssignments(String appId)
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
Collection<String> domains =

supApplication.getApplicationDomainAssignments("app1");
```

Add Packages to an Application

Adds packages to the specified application.

Syntax

```
void addApplicationPackages(String appID, String domain,
Collection<String> pkgs) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Add packages –**

```
String domain = "default";
Collection<String> pkgs = new ArrayList<String>();
pkgs.add("pkg1");
supApplication.addApplicationPackages("app1", domain, pkgs);
```

Remove Packages from an Application

Removes packages from the specified application.

Syntax

```
void removeApplicationPackages(String appID, String domain,
Collection<String> pkgs) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Removal –**

  ```
  String domain = "default";
  Collection<String> pkgs = new ArrayList<String>();
  pkgs.add("pkg1");
  supApplication.removeaddApplicationPackages("app1", domain,
  pkgs);
  ```

Retrieval of a List of Packages from an Application

Retrieves a list of packages from an application that satisfy the filter. The return result is paginated

Syntax

```
PaginationResult<ApplicationPackageVO>
getApplicationPackages(Application_FilterSortVO filter, Long offset,
Integer length) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

  ```
  Package_FilterSortVO filter = new Package_FilterSortVO();
  FilterExpression<APPLICATION_PACKAGE> expression1 = new
  FilterExpression<APPLICATION_PACKAGE>();
  expression1 = expression1.eq(APPLICATION_PACKAGE.APPLICATION_ID,
  "app1");
  filter.setFilterExpression(expression1);
  filter.setSortField(APPLICATION_PACKAGE.DOMAIN);
  filter.setSortOrder(SORT_ORDER.ASCENDING);
  ```

```
PaginationResult<ApplicationPackageVO> apps =
supApplication.getApplicationPackages(filter, 0l, 100);
```

Managing Application Connections

Use the `SUPApplication` interface to manage registration of application connections. Operations you can perform with this interface include:

- Retrieving a list of application connections
- Cloning application connections
- Registering or re-registering an application connection
- Updating application connection settings
- Deleting an application connection
- Locking or unlocking an application connection

Retrieve Application Connections

Retrieves a list of application connections that satisfy the given filter. The return result is paginated.

Syntax

```
PaginationResult<ApplicationConnectionVO>
getApplicationConnections(AppConnection_FilterSoftVO filter, Long
offset, Integer length) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
AppConnectionFilterSortVO filter = new
AppConnectionFilterSortVO();
FilterExpression<APPCONNECTION> fe = new FilterExpression<
APPCONNECTION >();
FilterExpression< APPCONNECTION > fe1 =
fe.eq(APPCONNECTION.DOMAIN, "default");
filter.setFilterExpression(fe1);
filter.setSortField(APPCONNECTION.APPLICATION_ID);
PaginationResult<ApplicationConnectionVO> result = app
        .getApplicationConnections(filter, 0L, 10);
for (ApplicationConnectionVO appConn : result.getItems()) {
    System.out.println(appConn.getId());
}
```

Cloning Application Connections

Registers an application connection by cloning an existing application connection.

Syntax

```
Collection<Integer> cloneApplicationConnections(Collection<Map>
cloneRequests, Map settings) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Clone application connection –**

```
AppConnectionCloneRequestVO accrvo = new
AppConnectionCloneRequestVO();
Map<APPCONNECTION_CLONE, Object> req1 = new
HashMap<APPCONNECTION_CLONE, Object>();
req1.put(APPCONNECTION_CLONE.EXISTING_NUMERIC_ID, "8");
req1.put(APPCONNECTION_CLONE.ACTIVATION_CODE, "345");
req1.put(APPCONNECTION_CLONE.EXPIRATION_HOUR, "3");
req1.put(APPCONNECTION_CLONE.USER_ID, "river");
accrvo.setRequest(req1);

Collection<AppConnectionCloneRequestVO> reqs = new
ArrayList<AppConnectionCloneRequestVO>();
reqs.add(accrvo);

AppConnectionSettingVO acsvo = new AppConnectionSettingVO();
Map<APPCONNECTION_SETTING_FIELD, Object> setting = new
HashMap<APPCONNECTION_SETTING_FIELD, Object>();
setting.put(APPCONNECTION_SETTING_FIELD.SECURITY_CONF, "admin2");
setting.put(APPCONNECTION_SETTING_FIELD.ALLOW_ROAMING, "true");
acsvo.setSetting(setting);
app.cloneApplicationConnections(reqs, acsvo);
```

Register an Application Connection

Registers a batch of application connections.

Syntax

```
Collection<Integer> registerApplicationConnections(templateName,
registrationRequests, Map settings) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Register application connection –**

```
AppConnectionRegistrationRequestVO acrrvo1 = new
AppConnectionRegistrationRequestVO();
AppConnectionRegistrationRequestVO acrrvo2 = new
AppConnectionRegistrationRequestVO();

Map<APPCONNECTION_REGISTRATION, Object> req1 = new
HashMap<APPCONNECTION_REGISTRATION, Object>();
req1.put(APPCONNECTION_REGISTRATION.USER_ID,
        contextFactory.getProperty("sup.app.user.1"));
req1.put(APPCONNECTION_REGISTRATION.ACTIVATION_CODE, "1234");
req1.put(APPCONNECTION_REGISTRATION.EXPIRATION_HOUR, "1");
acrrvo1.setRequest(req1);

Map<APPCONNECTION_REGISTRATION, Object> req2 = new
HashMap<APPCONNECTION_REGISTRATION, Object>();
req2.put(APPCONNECTION_REGISTRATION.USER_ID,
        contextFactory.getProperty("sup.app.user.2"));
req2.put(APPCONNECTION_REGISTRATION.ACTIVATION_CODE, "5678");
req2.put(APPCONNECTION_REGISTRATION.EXPIRATION_HOUR, "1");
acrrvo2.setRequest(req2);

Collection<AppConnectionRegistrationRequestVO> reqs = new
ArrayList<AppConnectionRegistrationRequestVO>();
reqs.add(acrrvo1);
reqs.add(acrrvo2);

AppConnectionSettingVO settings = new AppConnectionSettingVO();
Map<APPCONNECTION_SETTING_FIELD, Object> setting = new
HashMap<APPCONNECTION_SETTING_FIELD, Object>();
setting.put(APPCONNECTION_SETTING_FIELD.SECURITY_CONF,
        contextFactory.getProperty("sup.secconf.1"));
setting.put(APPCONNECTION_SETTING_FIELD.ALLOW_ROAMING, "true");
setting.put(APPCONNECTION_SETTING_FIELD.SERVER_NAME,
"localhost");
settings.setSetting(setting);
app.registerApplicationConnections(templateName, reqs, settings);
```

Re-register an Application Connection

Re-registers an application connection.

Syntax

```
Collection<Integer>
reregisterApplicationConnections(Collection<Map>
reregistrationRequests, Map settings) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Re-registration –**

```
AppConnectionReregistrationRequestVO acrrvo1 = new
AppConnectionReregistrationRequestVO();

Map<APPCONNECTION_REREGISTRATION, Object> req1 = new
HashMap<APPCONNECTION_REREGISTRATION, Object>();
req1.put(APPCONNECTION_REREGISTRATION.EXISTING_NUMERIC_ID, "5");
req1.put(APPCONNECTION_REREGISTRATION.ACTIVATION_CODE, "15");
req1.put(APPCONNECTION_REREGISTRATION.EXPIRATION_HOUR, "2");
req1.put(APPCONNECTION_REREGISTRATION.USER_ID, "hel");
acrrvo1.setRequest(req1);

Collection<AppConnectionReregistrationRequestVO> reqs = new
ArrayList<AppConnectionReregistrationRequestVO>();
reqs.add(acrrvo1);

AppConnectionSettingVO settings = new AppConnectionSettingVO();
Map<APPCONNECTION_SETTING_FIELD, Object> setting = new
HashMap<APPCONNECTION_SETTING_FIELD, Object>();
setting.put(APPCONNECTION_SETTING_FIELD.SERVER_NAME, "helxp-
vm1");
setting.put(APPCONNECTION_SETTING_FIELD.SERVER_PORT, "8888");
setting.put(APPCONNECTION_SETTING_FIELD.FARM_ID, "1");
setting.put(APPCONNECTION_SETTING_FIELD.DOMAIN, "default");
setting.put(APPCONNECTION_SETTING_FIELD.SECURITY_CONF, "admin1");
settings.setSetting(setting);
app.reregisterApplicationConnections(reqs, settings);
```

Application Connection Settings Update

Updates the settings of a list of application connections.

Syntax

```
void updateApplicationConnectionSettings(Collection<Integer>
numericIds, Map settings) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
PaginationResult<ApplicationConnectionVO> result = app
.getApplicationConnections(filter, 0L, NULL);
Collection<Integer> appConnIds = new ArrayList<Integer>();

for (ApplicationConnectionVO appConn : result.getItems()) {
appConnIds.add(appConn.getNumericId());
}
```

```
AppConnectionSettingVO settings = new AppConnectionSettingVO();
Map<APPCONNECTION_SETTING_FIELD, Object> setting = new
HashMap<APPCONNECTION_SETTING_FIELD, Object>();
setting.put(APPCONNECTION_SETTING_FIELD.SECURITY_CONF, "admin");
settings.setSetting(setting);
app.updateApplicationConnectionSettings(appConnIds, settings);
```

Application Connection Deletion

Deletes a list of application connections.

Syntax

```
void deleteApplicationConnections(Collection<Integer> numericIds)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Create registration template –** deletes the specified registration templates ("Default" and "testTemplate2"):

```
Collection<Integer> appConnIds = new ArrayList<Integer>();
appConnIds.add(7);
appConnIds.add(8);

app.deleteApplicationConnections(appConnIds);
```

Lock or Unlock Application Connection

Locks or unlocks a list of application connections.

Syntax

```
void lockApplicationConnections(Collection<String>
applicationConnectionIds) throws SUPAdminException;

void unlockApplicationConnections(Collection<String>
applicationConnectionIds) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Lock or Unlock Application Connection –**

```
PaginationResult<ApplicationConnectionVO> result =
app.getApplicationConnections(filter, 0L, NULL);
Collection<String> appConnIds = new ArrayList<String>();
```

```
for (ApplicationConnectionVO appConn : result.getItems()) {
  appConnIds.add(appConn.getId());
}

app.lockApplicationConnection(appConnIds);
app.unlockApplicationConnection(appConnIds);
```

Usage

This API requires the application connection ID of the application connection (and not the numeric ID of the application connection).

Managing Application Connection Templates

Use the `SUPApplication` interface to manage application connection templates. Operations you can perform with this interface include:

- Retrieving a list of application connection templates
- Creating an application connection template
- Updating application connection template settings
- Deleting an application connection template

Application Connection Template Retrieval

Retrieves a list of application connection templates.

Syntax

```
PaginationResult<ApplicationConnectionTemplateVO>
getApplicationConnectionTemplates(AppConnectionTemplateFilterSortVO
filter, Long offset, Integer length) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
AppConnectionTemplateFilterSortVO filter = new
AppConnectionTemplateFilterSortVO();
FilterExpression<APPCONNECTION_TEMPLATE> fe = new
FilterExpression<APPCONNECTION_TEMPLATE>();

FilterExpression<APPCONNECTION_TEMPLATE> fe1 =
fe.eq(APPCONNECTION_TEMPLATE.DOMAIN, "default");

FilterExpression<APPCONNECTION_TEMPLATE> fe2 =
fe.eq(APPCONNECTION_TEMPLATE.SECURITY_CONF, "admin");
```

```
fe = fe1.and(fe2);
filter.setFilterExpression(fe);
PaginationResult<ApplicationConnectionTemplateVO> result = app
        .getApplicationConnectionTemplates(filter, 0L, 10);
for (ApplicationConnectionTemplateVO appConnT :
result.getItems()) {
    System.out.println(appConnT.getName());
}
```

Application Connection Template Creation

Creates an application connection templates with the specified settings.

Syntax

```
void
createApplicationConnectionTemplate(ApplicationConnectionTemplateVO
applicationConnectionTemplate, Map settings) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Creation –**

```
AppConnectionSettingVO acsvo = new AppConnectionSettingVO();
Map<APPCONNECTION_SETTING_FIELD, Object> setting = new
HashMap<APPCONNECTION_SETTING_FIELD, Object>();
setting.put(APPCONNECTION_SETTING_FIELD.SECURITY_CONF, "admin");
acsvo.setSetting(setting);

app.createApplicationConnectionTemplate("MyTemplate",
        "Short description", acsvo);
```

Update of Application Connection Template Settings

Updates application connection template settings.

Syntax

```
void updateApplicationConnectionTemplateSettings(templateName, Map
settings) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
AppConnectionSettingVO settings = new AppConnectionSettingVO();
Map<APPCONNECTION_SETTING_FIELD, Object> setting = new
```

```
HashMap<APPCONNECTION_SETTING_FIELD, Object>();
setting.put(APPCONNECTION_SETTING_FIELD.SECURITY_CONF, "admin");
setting.put(APPCONNECTION_SETTING_FIELD.ACTIVATION_CODE_LENGTH,
"9");
setting.put(APPCONNECTION_SETTING_FIELD.ALLOW_ROAMING, "true");
settings.setSetting(setting);
app.updateApplicationConnectionTemplateSettings("template 1",
settings);
```

Application Connection Template Deletion

Deletes a list of application connection templates.

Syntax

```
void deleteApplicationConnectionTemplates(List<String>
templateNames) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

```
Collection<String> names = new ArrayList<String>();
names.add("MyTemplate");

app.deleteApplicationConnectionTemplates(names);
```

Monitoring Unwired Platform Components

`SUPMonitor` provides most of the operations related to monitoring of Sybase Unwired Platform components. `SUPCluster` provides additional operations.

Start Monitoring Management

Starts the management of an Unwired Server monitoring operations.

Syntax

```
public static SUPMonitor getSUPMonitor(ClusterContext
clusterContext) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start monitoring management –**

```
clusterContext = serverContext.getClusterContext("<cluster
name>");
SUPMonitor supMonitor =
SUPObjectFactory.getSUPMonitor(clusterContext);
```

Usage

To manage Unwired Server monitoring operations, you must create an instance of `SUPMonitor`.

Retrieval of Monitoring Profiles Using SUPCluster

Retrieves the monitoring profiles in a cluster.

Syntax

```
Collection<MonitoringProfileVO> getMonitoringProfiles() throws
SUPAdminException;

MonitoringProfileVO getMonitoringProfile(String name) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
Collection<MonitoringProfileVO> mpvos = supCluster
        .getMonitoringProfiles();
MonitoringProfileVO mpvo = supCluster
        .getMonitoringProfile("<monitoring configuration name>");
System.out.println(mpvo.getName());
```

Creation of a Monitoring Profile Using SUPCluster

Creates a monitoring profile in a cluster.

Syntax

```
void createMonitoringProfile(MonitoringProfileVO mpvo) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Create monitoring profile –**

```
//Create a monitoring profile
MonitoringProfileVO mpvo_new = new MonitoringProfileVO();
mpvo_new.setName("<monitoring configuration new name>");
mpvo_new.setDurationType(MONITORING_DURATION_TYPE.SCHEDULED);
mpvo_new.setEnabled(true);

MonitoredDomain md = new MonitoredDomain("<domain name>");
md.setName("<domain name>");
MonitoredPackage mp1 = new MonitoredPackage("<package name 1>");
MonitoredPackage mp2 = new MonitoredPackage("<package name 2>");
md.setMonitoredPackages(Arrays
        .asList(new MonitoredPackage[] { mp1, mp2 }));
mpvo_new.setMonitoredDomains(Arrays.asList(new MonitoredDomain[]
{ md }));

ScheduleVO svo = new ScheduleVO();
svo.setEndDate(new Date());
svo.setEndTime(new Date());
svo.setStartDate(new Date(0));
svo.setStartTime(new Date(0));
svo.setInterval(1234);
svo.setFreq(SCHEDULE_FREQ.INTERVAL);
EnumSet<DAY_OF_WEEK> dayofweeks =
EnumSet.noneOf(DAY_OF_WEEK.class);
svo.setDaysofweek(dayofweeks);
dayofweeks.add(DAY_OF_WEEK.MONDAY);
mpvo_new.setSchedule(svo);
supCluster.createMonitoringProfile(mpvo_new);
```

Update of a Monitoring Profile Using SUPCluster

Updates a monitoring profile in a cluster.

Syntax

```
void updateMonitoringProfile(MonitoringProfileVO monitoringProfile)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update monitoring profile –**

```
// Update monitoring profile
MonitoringProfileVO mpvo = supCluster
        .getMonitoringProfile("<monitoring configuration name>");
mpvo.getSchedule().setFreq(SCHEDULE_FREQ.INTERVAL);
```

```
mpvo.getSchedule().setInterval(200000);
supCluster.updateMonitoringProfile(mpvo);
```

Usage

A monitoring profile you create with this method replaces a profile with the same name on the Unwired Server.

Deletion of a Monitoring Profile Using SUPCluster

Deletes a monitoring profiles from a cluster.

Syntax

```
void deleteMonitoringProfile(String name) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Delete monitoring profile –**

```
// Delete monitoring profile
supCluster.deleteMonitoringProfile("<monitoring configuration
name>");
```

Deletion of Monitoring Data Using SUPCluster

Deletes monitoring data.

Syntax

```
void deleteMonitoringData(Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Delete monitoring data –** deletes monitoring data for the specified time period (between the `startTime` and the `endTime`):

```
Date startTime = new Date(0);
Date endTime = new Date();
supCluster.deleteMonitoringData(startTime, endTime);
```

Construct a Path to the Monitored Object

To retrieve monitoring data, you must provide an instance or collection of `MonitoredObject` to specify the data that gets returned.

`MonitoredObject` contains subclasses in this logical hierarchy:

- `MonitoredCluster`
  - `MonitoredDomain`
    - `MonitoredPackage`
      - `MonitoredSyncGroup`
      - `MonitoredCacheGroup`
        - `MonitoredMBO`
          - `MonitoredOperation`

With this hierarchy, an object can be identified using a path-like structure. Such a path acts as a context against which monitoring data is searched and returned. Follow these rules when constructing a path:

- Start with `MonitoredCluster`.
- Except for `MonitoredCluster`, if `Monitored*` appears in a path, then the class logically above it is in the path.
- `MonitoredSyncGroup` and `MonitoredCache` are mutual exclusive in a path.

Retrieval of a Large Volume of Monitoring Data

Retrieves a specified portion of a large volume of monitoring data (for example, user access histories).

Syntax

```
Long getSecurityLogHistoryCount(Collection<MonitoredObject>
monitoredObjects, Boolean accessResult, Date startTime, Date
endTime) throws SUPAdminException;

Collection<SecurityLogHistoryVO>
getSecurityLogHistory(Collection<MonitoredObject> monitoredObjects,
Boolean accessResult, Date startTime, Date endTime, Long offset,
Integer length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
mc.addMonitoredDomain(new MonitoredDomain("default"));
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });

long count = supMonitor.getSecurityLogHistoryCount(mos, null,
null, null);
Collection<SecurityLogHistoryVO> slhvos =
supMonitor.getSecurityLogHistory(mos, null,
        null, null, null, null, null);
for (SecurityLogHistoryVO slhvo : slhvos) {
    System.out.println(slhvo.getUserName());
}
long offset = slhvos.size();
while(offset<count){
    slhvos = supMonitor.getSecurityLogHistory(mos, null,
            null, null, offset, null, null);
    for (SecurityLogHistoryVO slhvo : slhvos) {
        System.out.println(slhvo.getUserName());
    }
    offset += slhvos.size();
```

Usage

When monitoring a large volume of data, a paginated API allows you to get a total row count for retrieving the data in chunks. `Offset` specifies where the returned data starts for this call. `Length` specifies the maximum number of records returned for this call.

Specify Result Sorting

You can specify an instance of SortedField to sort the returned result on the given field in the given order (ascending or descending).

Each type of monitoring data has a different set of sortable fields.

- Data change notification
  - DOMAIN
  - NOTIFICATION_TIME
  - PACKAGE
  - PROCESS_TIME
  - PUBLICATION
- Device notification

    - DEVICE_ID
    - DOMAIN
    - NOTIFICATION_TIME
    - PACKAGE
    - PUBLICATION
    - SUBSCRIPTION_ID
    - USER_NAME
- Messaging summary
    - DOMAIN_NAME
    - LAST_TIME_IN
    - LAST_TIME_OUT
    - PACKAGE
    - SUBSCRIPTION_COMMAND_COUNT
    - TOTAL_ERRORS
    - TOTAL_MESSAGES_RECEIVED
    - TOTAL_MESSAGES_SENT
    - TOTAL_OPERATION_REPLAYS
    - TOTAL_PAYLOAD_RECEIVED
    - TOTAL_PAYLOAD_SENT
- Messaging details
    - DEVICE
    - DOMAIN_NAME
    - ERROR
    - FINISH_TIME
    - MBO
    - MESSAGE_TYPE
    - OPERATION_NAME
    - PACKAGE
    - PAYLOAD_SIZE
    - PROCESS_TIME
    - START_TIME
    - USER
- Replication summary
    - DOMAIN_NAME
    - PACKAGE
    - START_TIME
    - SYNC_TIME
    - TOTAL_BYTES_RECEIVED
    - TOTAL_BYTES_SENT

    - TOTAL_ERRORS
    - TOTAL_OPERATION_REPLAYS
    - TOTAL_ROWS_SENT
- Replication details
    - BYTES_TRANSFERRED
    - DEVICE
    - DOMAIN_NAME
    - ERROR
    - FINISH_TIME
    - OPERATION_NAME
    - OPERATION_NAME
    - PACKAGE
    - START_TIME
    - SYNC_PHASE
    - TOTAL_BYTES_SENT
    - TOTAL_ROWS_SENT
    - USER
- Security access
    - DEVICE_ID
    - DOMAIN
    - OUTCOME
    - PACKAGE
    - SECURITY_CONFIGURATION
    - TIME
    - USER

Retrieval of Security Log History

Retrieves a security log history for specified monitored objects, determines how many records are available, and specifies how to retrieve and sort the data.

Syntax

```
Long getSecurityLogHistoryCount(Collection<MonitoredObject>
monitoredObjects, Boolean accessResult, Date startTime, Date
endTime) throws SUPAdminException;

Collection<SecurityLogHistoryVO>
getSecurityLogHistory(Collection<MonitoredObject> monitoredObjects,
Boolean accessResult, Date startTime, Date endTime, Long offset,
Integer length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
// Prepare monitored objects
MonitoredCluster mc = new MonitoredCluster();
mc.addMonitoredDomain(new MonitoredDomain("default"));
mc.addMonitoredDomain(new MonitoredDomain("test"));
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });

// Prepare time range
Date startTime = new Date(0);
Date endTime = new Date();

// Should only return successful access
Boolean accessResult = true;

// Starting from 10th record
Long offset = 10L;
// Try to retrieve 10000 records
Integer target = 10000;

// Specify sorting field and sorting order
SortedField<SortedField.SECURITY_ACCESS> sf = new
SortedField<SortedField.SECURITY_ACCESS>(
        SECURITY_ACCESS.DOMAIN, SORT_ORDER.ASCENDING);

// See how many records are available
long count = supMonitor.getSecurityLogHistoryCount(mos,
accessResult,
        startTime, endTime);
long available = Math.min(count - offset, target);
if (available < 1) {
    System.out.println("No monitoring data found at offset " +
offset);
    return;
} else {
    System.out.println("There " + available
            + " records monitoring data at offset " + offset);
}

// Specify the preferred record number to be fetched from server
in one
// call.
// Management server has imposed a upper limit of 500 for sake of
// performance.
Integer length = new Integer(new Long(Math.min(500, available))
        .intValue());
Collection<SecurityLogHistoryVO> slhvos =
supMonitor.getSecurityLogHistory(mos,
```

```
        accessResult, startTime, endTime, offset, length, sf);
// All the available records can be fetched at one call.
if (slhvos.size() == available) {
    System.out.println("Fetched " + available + " of " + available
            + " records of monitoring data.");
    return;
}
long read = slhvos.size();
offset += read;
while (read < available) {
    slhvos = supMonitor.getSecurityLogHistory(mos, accessResult,
            startTime, endTime, offset, length, sf);
    System.out.println("Fetched " + slhvos.size() + " of " +
available
            + " records of monitoring data.");
    read += slhvos.size();
    offset += read;
}
```

Retrieval of Current Messaging Requests

Retrieves current messaging requests for the specified domains and packages.

Syntax

```
Collection<MessagingRequestVO>
getMessagingRequests(Collection<MonitoredObject> monitoredObjects)
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
for (MessagingRequestVO mrvo :
supMonitor.getMessagingRequests(mos)) {
    System.out.println(mrvo.getPackageName());
}
```

Retrieval of Detailed Messaging History

Retrieves a detailed messaging history for the specified domains and packages.

Syntax

```
Collection<MessagingHistoryDetailVO>
getMessagingHistoryDetail(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime, Long offset, Integer
length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves a detailed messaging history for the specified domains and packages (the "test_mbs:1.0" and "test_mbs:2.0" packages from the "default" domain, and the "test_mbs:3.0" and "test_mbs:4.0" packages from the "test" domain):

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
System.out.println(supMonitor.getMessagingHistoryDetail(mos,
null, null, null, null, null));
```

Note: See *Developer Guide for Unwired Server Management API > Code Samples > Monitoring Unwired Platform Components > Retrieval of a Large Volume of Monitoring Data* for handling the large volume of data that this method may retrieve.

Retrieval of Summary Messaging History

Retrieves a summary of the messaging history for the specified domains and packages.

Syntax

```
Collection<MessagingHistorySummaryVO>
getMessagingHistorySummary(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime, Long offset, Integer
length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves a summary of the messaging history for the specified domains and packages (the "test_mbs:1.0" and "test_mbs:2.0" packages from the "default" domain, and the "test_mbs:3.0" and "test_mbs:4.0" packages from the "test" domain):

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
System.out.println(supMonitor.getMessagingHistorySummary(mos,
null, null, null, null, null));
```

Note: See *Developer Guide for Unwired Server Management API > Code Samples > Monitoring Unwired Platform Components > Retrieval of a Large Volume of Monitoring Data* for handling the large volume of data that this method may retrieve.

Messaging Performance Retrieval

Retrieves the messaging performance data for the specified domains and packages.

Syntax

```
MessagingPerformanceVO
getMessagingPerformance(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves the messaging performance data for the specified domains and packages (the "test_mbs:1.0" and "test_mbs:2.0" packages from the "default" domain, and the "test_mbs:3.0" and "test_mbs:4.0" packages from the "test" domain):

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
```

```
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
MessagingPerformanceVO mpvo =
supMonitor.getMessagingPerformance(mos,
        null, null);
System.out.println(mpvo.getMboForMaxProcessTime());
```

Messaging Statistics Retrieval

Retrieves the messaging statistics for a cluster, a domain, a package, or a specific mobile business object.

Syntax

```
MessagingStatisticsVO getMessagingStatistics(MonitoredObject
monitoredObject, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Cluster-level messaging statistics –** retrieves the messaging statistics for all domains in a cluster:

  ```
  MonitoredCluster mc = new MonitoredCluster();

  // Retrieve cluster-level messaging statistics (statistics for all
  domains).
  supMonitor.getMessagingStatistics(mc, null, null);
  ```

- **Domain-level messaging statistics –** retrieves the messaging statistics for all packages in a domain:

  ```
  MonitoredCluster mc = new MonitoredCluster();
  MonitoredDomain md = new MonitoredDomain("default");

  // Retrieve domain-level messaging statistics (statistics for all
  packages).
  mc.addMonitoredDomain(md);
  supMonitor.getMessagingStatistics(mc, null, null);
  ```

- **Package-level messaging statistics –** retrieves the messaging statistics for all MBOs in a package:

  ```
  MonitoredCluster mc = new MonitoredCluster();
  MonitoredDomain md = new MonitoredDomain("default");
  ```

```
MonitoredPackage mp = new MonitoredPackage("test_mbs:1.0");

// Retrieve package-level messaging statistics (statistics for all
MBOs).
md.addMonitoredPackage(mp);
supMonitor.getMessagingStatistics(mc, null, null);
```

- **MBO messaging statistics –** retrieves the messaging statistics for a specific mobile business object:

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md = new MonitoredDomain("default");
MonitoredPackage mp = new MonitoredPackage("test_mbs:1.0");
// Monitored cache does not contribute to messaging statistics,
but in
// order to retain the validity of the monitored object path, it
should be
// part of the path.
MonitoredCacheGroup mcg = new MonitoredCacheGroup("Default");
MonitoredMBO mmbo = new MonitoredMBO("Customer");

// Retrieve messaging statistics for a specific MBO.
mcg.addMonitoredMBO(mmbo);
supMonitor.getMessagingStatistics(mc, null, null);
```

Retrieval of Current Replication Requests

Retrieves current replication requests for the specified domains and packages.

Syntax

```
Collection<ReplicationRequestVO>
getReplicationRequests(Collection<MonitoredObject>
monitoredObjects) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
```

```
        .asList(new MonitoredObject[] { mc });
System.out.println(supMonitor.getReplicationRequests(mos));
```

Retrieval of Detailed Replication History

Retrieves a detailed replication history for the specified domains and packages.

Syntax

```
Collection<ReplicationHistoryDetailVO>
getReplicationHistoryDetail(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime, Long offset, Integer
length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
System.out.println(supMonitor.getReplicationHistoryDetail(mos,
null, null, null, null, null));
```

Retrieval of Summary Replication History

Retrieves a summary of replication history for the specified domains and packages.

Syntax

```
Collection<ReplicationHistorySummaryVO>
getReplicationHistorySummary(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime, Long offset, Integer
length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
System.out.println(supMonitor.getReplicationHistorySummary(mos,
null, null, null, null, null));
```

Replication Performance Retrieval

Retrieves replication performance data for the specified domains and packages.

Syntax

```
ReplicationPerformanceVO
getReplicationPerformance(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
ReplicationPerformanceVO rpvo =
supMonitor.getReplicationPerformance(mos, null, null);
System.out.println(rpvo.getMaxSyncTime());
```

Replication Statistics Retrieval

Retrieves the replication statistics for a cluster, a domain, a package, or a specific mobile business object.

Syntax

```
ReplicationStatisticsVO getReplicationStatistics(MonitoredObject
monitoredObject, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Cluster-level replication statistics –** retrieves the replication statistics for all domains in a cluster:

  ```
  MonitoredCluster mc = new MonitoredCluster();

  //Retrieve cluster-level replication statistics (for all domains).
  supMonitor.getReplicationStatistics(mc, null, null);
  ```

- **Domain-level replication statistics –** retrieves the replication statistics for all packages in a domain:

  ```
  MonitoredCluster mc = new MonitoredCluster();
  MonitoredDomain md = new MonitoredDomain("default");

  //Retrieve domain-level replication statistics (for all packages).
  mc.addMonitoredDomain(md);
  supMonitor.getReplicationStatistics(mc, null, null);
  ```

- **Package-level replication statistics –** retrieves the replication statistics for all MBOs in a package:

  ```
  MonitoredCluster mc = new MonitoredCluster();
  MonitoredDomain md = new MonitoredDomain("default");
  MonitoredPackage mp = new MonitoredPackage("test_mbs:1.0");

  //Retrieve package-level replication statistics (for all MBOs) .
  md.addMonitoredPackage(mp);
  supMonitor.getReplicationStatistics(mc, null, null);
  ```

- **MBO replication statistics –** retrieves the replication statistics for a specific mobile business object:

  ```
  MonitoredCluster mc = new MonitoredCluster();
  MonitoredDomain md = new MonitoredDomain("default");
  MonitoredPackage mp = new MonitoredPackage("test_mbs:1.0");
  // Monitored cache does not contribute to replication statistics,
  however
  // to retain the validity of the monitored object path, it should
  ```

```
be part of the path.
MonitoredCacheGroup mcg = new MonitoredCacheGroup("Default");
MonitoredMBO mmbo = new MonitoredMBO("Customer");

//Retrieve replication statistics for a specific MBO.
mcg.addMonitoredMBO(mmbo);
supMonitor.getReplicationStatistics(mc, null, null);
```

Retrieval of Data Change Notification History

Retrieves data change notification history for a monitored cluster.

Syntax

```
Collection<DataChangeNotificationHistoryVO>
getDataChangeNotificationHistory(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime, Long offset, Integer
length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
mc.addMonitoredDomain(new MonitoredDomain("default"));
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
System.out.println(supMonitor.getDataChangeNotificationHistory(mo
s,
null, null, null, null, null));
```

Retrieval of Data Change Notification Performance

Retrieves data change notification performance for monitored objects in a cluster.

Syntax

```
DataChangeNotificationPerformanceVO
getDataChangeNotificationPerformance(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
MonitoredCluster mc = new MonitoredCluster();
mc.addMonitoredDomain(new MonitoredDomain("default"));
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
DataChangeNotificationPerformanceVO npvo = supMonitor
        .getDataChangeNotificationPerformance(mos, null, null);
System.out.println(npvo.getMinProcessingTime());
```

Retrieval of Device Notification History

Retrieves device notification history for the monitored objects in a cluster.

Syntax

```
Collection<DeviceNotificationHistoryVO>
getDeviceNotificationHistory(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime, Long offset, Integer
length, SortedField<? extends Enum> sortedField) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves device notification history for the "default" domain in a cluster:

```
MonitoredCluster mc = new MonitoredCluster();
mc.addMonitoredDomain(new MonitoredDomain("default"));
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
System.out.println(supMonitor.getDeviceNotificationHistory(mos,
null, null, null, null, null));
```

Retrieval of Device Notification Performance

Retrieves device notification performance for the monitored objects in a cluster.

Syntax

```
DeviceNotificationPerformanceVO
getDeviceNotificationPerformance(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves device notification performance for the monitored "default" domain in a cluster:

```
MonitoredCluster mc = new MonitoredCluster();
mc.addMonitoredDomain(new MonitoredDomain("default"));
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
DeviceNotificationPerformanceVO dnpvo = supMonitor
        .getDeviceNotificationPerformance(mos, null, null);
System.out.println(dnpvo.getDistinctDevices());
```

Retrieval of Cache Group Performance

Retrieves cache group perfomance data of the monitored objects within a specified time range.

Syntax

```
Collection<CacheGroupPerformanceVO>
getCacheGroupPerformance(Collection<MonitoredObject>
monitoredObjects, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves cache group perfomance data for the specified domains and packages:

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredDomain md_tst = new MonitoredDomain("test");
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:1.0"));
md_def.addMonitoredPackage(new MonitoredPackage("test_mbs:2.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:3.0"));
md_tst.addMonitoredPackage(new MonitoredPackage("test_mbs:4.0"));
mc.addMonitoredDomain(md_def);
mc.addMonitoredDomain(md_tst);
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
for (CacheGroupPerformanceVO cpvo : supMonitor
        .getCacheGroupPerformance(mos, null, null)) {
```

```
    System.out.println(cpvo.getMaxCacheHits());
}
```

Retrieval of Cache Group Statistics

Retrieves cache group statistics for a package or for an MBO within the specified time range.

Syntax

```
Collection<CacheGroupPackageStatisticsVO>
getCacheGroupPackageStatistics(MonitoredObject monitoredObject, Date
startTime, Date endTime) throws SUPAdminException;

Collection<CacheGroupMBOStatisticsVO>
getCacheGroupMBOStatistics(MonitoredObject monitoredObject, Date
startTime, Date endTime) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Package –** retrieves cache group statistics for the specified package in a domain:

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
MonitoredPackage mp = new MonitoredPackage("jdbc:1.0");
md_def.addMonitoredPackage(mp);
mc.addMonitoredDomain(md_def);
for (CacheGroupPackageStatisticsVO cgpsvo : supMonitor
        .getCacheGroupPackageStatistics(mc, null, null)) {
    System.out.println(cgpsvo.getRowCount());
}

mp.addMonitoredCacheGroup(new MonitoredCacheGroup("default"));
for (CacheGroupPackageStatisticsVO cgpsvo : supMonitor
        .getCacheGroupPackageStatistics(mc, null, null)) {
    System.out.println(cgpsvo.getRowCount());
}
```

- **MBO –** retrieves cache group statistics for the specified package, cache group, and MBO:

```
MonitoredCluster mc = new MonitoredCluster();
MonitoredDomain md_def = new MonitoredDomain("default");
mc.addMonitoredDomain(md_def);
for (CacheGroupMBOStatisticsVO cgmsvo : supMonitor
        .getCacheGroupMBOStatistics(mc, null, null)) {
    System.out.println(cgmsvo.getAccessCount());
}

MonitoredPackage mp = new MonitoredPackage("jdbc:1.0");
md_def.addMonitoredPackage(mp);
for (CacheGroupMBOStatisticsVO cgmsvo : supMonitor
        .getCacheGroupMBOStatistics(mc, null, null)) {
```

```
    System.out.println(cgmsvo.getAccessCount());
}

MonitoredCacheGroup mcg = new MonitoredCacheGroup("default");
mp.addMonitoredCacheGroup(mcg);
for (CacheGroupMBOStatisticsVO cgmsvo : supMonitor
        .getCacheGroupMBOStatistics(mc, null, null)) {
    System.out.println(cgmsvo.getAccessCount());
}

MonitoredMBO mmbo = new MonitoredMBO("Customer");
mcg.addMonitoredMBO(mmbo);
for (CacheGroupMBOStatisticsVO cgmsvo : supMonitor
        .getCacheGroupMBOStatistics(mc, null, null)) {
    System.out.println(cgmsvo.getAccessCount());
}
```

Retrieval of Queue Monitoring Data and Statistics

Retrieves a list of the monitoring statistics of Java Message Service (JMS) queues of the Unwired Server within the specified time range.

Syntax

```
Collection<MessagingQueueStatisticsVO>
getMessagingQueueStatistics(Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
for (MessagingQueueStatisticsVO mqsvo : supMonitor
        .getMessagingQueueStatistics(null, null)) {
    System.out.println(mqsvo.getQueueName());
}
```

Monitoring Data Export

Export access history of the monitored objects during the specified time range.

Exporting monitoring data is similar to retrieving monitoring data, with these differences:

- Exporting monitoring data requries an instance of `java.io.File`.
- You specify length to set the number of rows of records to be exported to a specified file. There is no server-side limitation on length.

Syntax

```
void exportSecurityLogHistory(File file, Collection<MonitoredObject>
monitoredObjects, Boolean accessResult, Date startTime, Date
endTime, Long offset, Integer length, SortedField<? extends Enum>
sortedField) throws SUPAdminException;

void exportMessagingQueueStatistics(File file, Date startTime, Date
endTime) throws SUPAdminException;

void exportMessagingRequests(File file, Collection<MonitoredObject>
monitoredObjects) throws SUPAdminException;

void exportMessagingHistorySummary(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime, Long offset, Integer length, SortedField<? extends Enum>
sortedField) throws SUPAdminException;

void exportMessagingHistoryDetail(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime, Long offset, Integer length, SortedField<? extends Enum>
sortedField) throws SUPAdminException;

void exportMessagingPerformance(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime) throws SUPAdminException;

void exportMessagingStatistics(File file, String user, Date
startTime, Date endTime) throws SUPAdminException;

void exportReplicationRequests(File file,
Collection<MonitoredObject> monitoredObjects) throws
SUPAdminException;

void exportReplicationHistorySummary(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime, Long offset, Integer length, SortedField<? extends Enum>
sortedField) throws SUPAdminException;

void exportReplicationHistoryDetail(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime, Long offset, Integer length, SortedField<? extends Enum>
sortedField) throws SUPAdminException;

void exportReplicationPerformance(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime) throws SUPAdminException;

void exportReplicationStatistics(File file, MonitoredObject
monitoredObject, Date startTime, Date endTime) throws
SUPAdminException;

void exportOperationStatistics(File file, MonitoredObject
monitoredObject, Date startTime, Date endTime) throws
SUPAdminException;
```

```
void exportDataChangeNotificationHistory(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime, Long offset, Integer length, SortedField<? extends Enum>
sortedField) throws SUPAdminException;

void exportDataChangeNotificationPerformance(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime) throws SUPAdminException;

void exportDeviceNotificationHistory(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime, Long offset, Integer length, SortedField<? extends Enum>
sortedField) throws SUPAdminException;

void exportDeviceNotificationPerformance(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime) throws SUPAdminException;

void exportCacheGroupPerformance(File file,
Collection<MonitoredObject> monitoredObjects, Date startTime, Date
endTime) throws SUPAdminException;

void exportCacheGroupPackageStatistics(File file, MonitoredObject
monitoredObject, Date startTime, Date endTime) throws
SUPAdminException;

void exportCacheGroupMBOStatistics(File file, MonitoredObject
monitoredObject, Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Export Security Log History –** exports records for a monitored domain to `access.log`:

```
File file = new File("D:\\tmp\\access.log");
MonitoredCluster mc = new MonitoredCluster();
mc.addMonitoredDomain(new MonitoredDomain("default"));
Collection<MonitoredObject> mos = Arrays
        .asList(new MonitoredObject[] { mc });
// when the method returns, the access.log contains the exported
records.
supMonitor.exportSecurityLogHistory(file, mos, null, null, null,
        null, null, null);
```

Managing Unwired Server Logs

You can enable logging and change log settings through the `SUPServerLog` interface. Operations you can perform with this interface include:

- Starting administration of logging.
- Constructing filters for a log.
- Filtering and retrieving log entries.
- Deleting a log.
- Managing log settings.

Start Log Management

Starts the management of logging for an Unwired Server.

Syntax

```
public static SUPServerLog getSUPServerLog(ServerContext
serverContext);
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start log management –**

  ```
  SUPServerLog supServerLog =
  SUPObjectFactory.getSUPServerLog(serverContext);
  ```

Usage

When an instance of `SUPServerLog` is returned from the `SUPObjectFactory`, call its method.

Log Filter Construction

You can define and compose filters to form a log fetching pattern. All the filters are subclasses of `FieldFilter`. There are two types of filters: those that act directly on log fields, and those that connect other filters.

These are the supported filters in `FieldFilter` for server logging:

- Direct Field Filters
  - FieldEqualityFilter

  - FieldRangeFilter
  - FieldRegexpFilter
  - FieldSetFilter
  - FieldWildcardFilter
- Connecting Filters
  - LogicalAndFilter
  - LogicalNotFilter
  - LogicalOrFilter

You cannot directly instantiate filters through a `new` operator. You must acquire them by calling methods of `SUPServerLog`.

```
FieldEqualityFilter bucket_eq = supServerLog.getFieldEqualityFilter(
        SERVER_LOG_FIELD.BUCKET, "MMS");

FieldSetFilter thread_set = supServerLog.getFieldSetFilter(
        SERVER_LOG_FIELD.THREAD_NAME, Arrays.asList(new String[] {
                "main", "dispatcher" }));

FieldWildcardFilter logger_wild =
supServerLog.getFieldWildcardFilter(
        SERVER_LOG_FIELD.LOGGER_NAME, "com.sybase.sup*");

FieldRangeFilter time_range = supServerLog.getFieldRangeFilter(
        SERVER_LOG_FIELD.TIMESTAMP, new Date(0), new Date());
FieldRegexpFilter regexp = supServerLog.getFieldRegexpFilter(
        SERVER_LOG_FIELD.THREAD_NAME, "^RMI");

LogicalNotFilter notFilter = supServerLog
        .getLogicalNotFilter(bucket_eq);
LogicalOrFilter orFilter = supServerLog.getLogicalOrFilter(Arrays
        .asList(new FieldFilter[] { time_range, regexp }));
LogicalAndFilter andFilter = supServerLog.getLogicalAndFilter(Arrays
        .asList(new FieldFilter[] { thread_set, logger_wild }));

FieldFilter filter = supServerLog.getLogicalAndFilter(Arrays
        .asList(new FieldFilter[] { notFilter, orFilter,
andFilter }));

supServerLog.setLogFilter(filter);
```

Log Entry Retrieval

Filters and retrieves entries from an Unwired Server log.

Syntax

```
void setLogPosition(LogPositionVO logPosition) throws
SUPAdminException;

Collection<LogEntryVO> getLogEntries(Integer start, Integer end)
throws SUPAdminException;
```

```
Collection<LogEntryVO> getLogEntries(Integer start, Integer end,
Boolean includingBackup) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Filter from the start of a log –** returns log entries from the start of the log (the 100th through 250th entries after the start of the log):

  ```
  supServerLog.setLogPosition(LogPositionVO.START);
  for (LogEntryVO levo : supServerLog.getLogEntries(100, 250)) {
      System.out.println(levo.getBucket());
  }
  for (LogEntryVO levo : supServerLog.getLogEntries(100, 250, true))
  {
      System.out.println(levo.getBucket());
  }
  ```

- **Filter from the end of a log –** returns log entries from the end of the log (the 100th to 250th entries before the end of the log):

  ```
  supServerLog.setLogPosition(LogPositionVO.END);
  for (LogEntryVO levo : supServerLog.getLogEntries(-100, -250)) {
      System.out.println(levo.getBucket());
  }
  for (LogEntryVO levo : supServerLog.getLogEntries(-100, -250,
  true)) {
      System.out.println(levo.getBucket());
  }
  ```

Log Deletion

Truncates a server log.

Syntax

```
void deleteLog() throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

  ```
  supServerLog.deleteLog();
  ```

Managing Log Settings

Sybase Unwired Platform server log settings are managed through metadata-based configuration and consist of one or more log appenders. Each log appender has one or more log buckets. They are represented by `LogAppenderVO` and `LogBucketVO` respectively.

These rules apply when managing server log settings:

- Each instance of `SUPServerLog` is a local object that holds values for all metadata based configuration. All of its methods perform against those values. The values are refreshed when `commit()` and `refresh()` are called.
- After getting an instance of `SUPServerLog`, call `refresh()` to populate the values, before calling any other methods.
- Changes made through these methods are cached locally unless you call the `commit()` method. `Commit()` sends all cached values (changed or not) to Unwired Server.

Populate Server Log Configuration

Populates the server log configuration values to Unwired Server.

Syntax

```
ConfigurationValidationStatus refresh() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Populate server log configuration –**

  ```
  supServerLog.refresh();
  ```

LogAppenderVO and LogBucketVO

The `LogAppenderVO` and `LogBucketVO` classes have two read-only properties that you must initialize at construction time.

- **ID –** a unique ID within the locally cached log configuration.
- **Type –** specifies the type of appender or bucket. The types of appenders and buckets are described in *Developer Guide for Unwired Server Management API > Client Metadata > Server Log Configuration*.

Retrieval of a List of Active Log Appenders

Retrieves a list of active log appenders.

Syntax

```
Collection<LogAppenderVO> getActiveLogAppenders() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

```
supServerLog.refresh();
for(LogAppenderVO lavo: supServerLog.getActiveLogAppenders()){
    System.out.println(lavo.getType());
    System.out.println(lavo.getProperties());
}
```

Update of an Active Log Appender

Updates an active log appender.

Syntax

```
void updateActiveLogAppender(String logAppenderID, LogAppenderVO
logAppender) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supServerLog.refresh();
LogAppenderVO lavo =
supServerLog.getActiveLogAppenders().iterator().next();
LogAppenderVO lavo_new = new LogAppenderVO(lavo.getID(),
lavo.getType());
Map<String, String> properties = new HashMap<String, String>();
properties.put("async", "true");
lavo_new.setProperties(properties);
supServerLog.updateActiveLogAppender(lavo_new.getID(), lavo_new);
supServerLog.commit();
```

Retrieval of a List of Active Log Buckets

Retrieves a list of active log buckets.

Syntax

```
Collection<LogAppenderVO> getActiveLogAppenders() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieve Active Log Buckets –**

```
supServerLog.refresh();
LogAppenderVO lavo =
supServerLog.getActiveLogAppenders().iterator().next();
for(LogBucketVO lbvo : lavo.getChildren()){
    System.out.println(lbvo.getType());
    System.out.println(lbvo.getProperties());
}
```

Update of an Active Log Bucket

Updates an active log bucket of an active log appender with the specified properties.

Syntax

```
void updateActiveLogBucket(String logAppenderID, String logBucketID,
LogBucketVO logBucket) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supServerLog.refresh();
LogAppenderVO lavo =
supServerLog.getActiveLogAppenders().iterator().next();
LogBucketVO lbvo = lavo.getChildren().iterator().next();
LogBucketVO lbvo_new = new LogBucketVO(lbvo.getID(),
lbvo.getType());
Map<String, String> properties = new HashMap<String, String>();
properties.put("LogLevel", "INFO");
lbvo_new.setProperties(properties);
supServerLog.updateActiveLogBucket (lavo.getID(),
lbvo_new.getID(), lbvo_new);
supServerLog.commit();
```

Managing Domain Logs

You can define log filtering and fetching behavior and change log settings for a domain through the `SUPDomainLog` interface.

Start Managing Domain Logs

Starts the management of logging for a domain.

Syntax

```
public static SUPDomainLog getSUPDomainLog(DomainContext
domainContext);
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Security configuration –**

  ```
  SUPDomainLog domainLog =
  SUPObjectFactory.getSUPDomainLog(domainContext);
  ```

Usage

When an instance of `SUPDomainLog` is returned from the `SUPObjectFactory`, call its method.

Retrieval of a List of Log Profiles

Retrieves a list of log profiles.

Syntax

```
Collection<DomainLogProfileVO> getDomainLogProfiles() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
for (DomainLogProfileVO dlpvo : domainLog.getDomainLogProfiles())
{
System.out.println(dlpvo.getName());
}
```

Creation of a Log Profile

Creates a log profile.

Syntax

```
void createDomainLogProfile(String profileName, String description,
Collection<DomainLogTrapVO> traps,
            Boolean enable) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **–**

```
String profileName = “profile1”;
Collection<DomainLogTrapVO<? extends Enum>> traps = new
ArrayList<DomainLogTrapVO<? extends Enum>>();
DomainLogTrapVO<DOMAIN_LOG_PROFILE_PACKAGE_TRAP> trap1 = new
DomainLogTrapVO<DOMAIN_LOG_PROFILE_PACKAGE_TRAP>(
DOMAIN_LOG_PROFILE_PACKAGE_TRAP.APPLICATION_ID);

DomainLogTrapVO<DOMAIN_LOG_PROFILE_SECURITY_TRAP> trap2 = new
DomainLogTrapVO<DOMAIN_LOG_PROFILE_SECURITY_TRAP>(
DOMAIN_LOG_PROFILE_SECURITY_TRAP.SECURITY_CONF);

DomainLogTrapVO<DOMAIN_LOG_PROFILE_ENDPOINT_TRAP> trap3 = new
DomainLogTrapVO<DOMAIN_LOG_PROFILE_ENDPOINT_TRAP>(
DOMAIN_LOG_PROFILE_ENDPOINT_TRAP.ENDPOINT);

DomainLogTrapVO<DOMAIN_LOG_PROFILE_APPCONNECTION_TRAP> trap4 =
new DomainLogTrapVO<DOMAIN_LOG_PROFILE_APPCONNECTION_TRAP>(
DOMAIN_LOG_PROFILE_APPCONNECTION_TRAP.APPLICATION_CONNECTION_ID);

DomainLogTrapVO<DOMAIN_LOG_PROFILE_PAYLOAD_TRAP> trap5 = new
DomainLogTrapVO<DOMAIN_LOG_PROFILE_PAYLOAD_TRAP>(
DOMAIN_LOG_PROFILE_PAYLOAD_TRAP.PAY_LOAD);

trap1.setEnabled(true);
trap1.setValues(Arrays.asList(new String[] { "app1:1.0",
"app2:2.0" }));

trap2.setEnabled(true);
```

```
trap2.setValues(Arrays.asList(new String[] { "admin", "test" }));

trap3.setEnabled(true);
EndpointTrapVO etvo1 = new EndpointTrapVO();
etvo1.setName("sampledb");
etvo1.setType(ENDPOINT_TYPE.JDBC);
EndpointTrapVO etvo2 = new EndpointTrapVO();
etvo2.setName("sap_crm:1.0");
etvo2.setType(ENDPOINT_TYPE.DOEC);
trap3.setValues(Arrays.asList(new EndpointTrapVO[] { etvo1,
etvo2 }));

trap4.setEnabled(true);
trap4.setValues(Arrays.asList(new String[] { "emulator1",
"bb2" }));

trap5.setEnabled(true);
trap5.setValues(Arrays
               .asList(new DOMAIN_LOG_CATEGORY[] {
                       DOMAIN_LOG_CATEGORY.DATA_SYNC,
                       DOMAIN_LOG_CATEGORY.GENERAL_DCN }));

traps.add(trap1);
traps.add(trap2);
traps.add(trap3);
traps.add(trap4);
traps.add(trap5);

domainLog.createDomainLogProfile(profileName, description, traps,
           false);
```

Update of a Log Profile

Updates a log profile.

Syntax

```
void updateDomainLogProfile(String profileName, String description,
Collection<DomainLogTrapVO> traps) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- –

```
String profileName = ”profile1”;
String description = "domain log profile description updated";

Collection<DomainLogTrapVO<? extends Enum>> traps = new
ArrayList<DomainLogTrapVO<? extends Enum>>();
DomainLogTrapVO<DOMAIN_LOG_PROFILE_PACKAGE_TRAP> trap1 = new
```

```
DomainLogTrapVO<DOMAIN_LOG_PROFILE_PACKAGE_TRAP>(
DOMAIN_LOG_PROFILE_PACKAGE_TRAP.APPLICATION_ID);

DomainLogTrapVO<DOMAIN_LOG_PROFILE_SECURITY_TRAP> trap2 = new
DomainLogTrapVO<DOMAIN_LOG_PROFILE_SECURITY_TRAP>(
DOMAIN_LOG_PROFILE_SECURITY_TRAP.SECURITY_CONF);

DomainLogTrapVO<DOMAIN_LOG_PROFILE_ENDPOINT_TRAP> trap3 = new
DomainLogTrapVO<DOMAIN_LOG_PROFILE_ENDPOINT_TRAP>(
DOMAIN_LOG_PROFILE_ENDPOINT_TRAP.ENDPOINT);

DomainLogTrapVO<DOMAIN_LOG_PROFILE_APPCONNECTION_TRAP> trap4 =
new DomainLogTrapVO<DOMAIN_LOG_PROFILE_APPCONNECTION_TRAP>(
DOMAIN_LOG_PROFILE_APPCONNECTION_TRAP.APPLICATION_CONNECTION_ID);

trap1.setEnabled(true);
trap2.setEnabled(true);
trap3.setEnabled(true);
trap4.setEnabled(true);

traps.add(trap1);
traps.add(trap2);
traps.add(trap3);
traps.add(trap4);

domainLog.updateDomainLogProfile(profileName, description,
traps);
```

Deletion of a Log Profile

Deletes a log profile.

Syntax

```
void deleteDomainLogProfiles(Collection<String> profileNames) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- –

```
String profileName = ”profile1”;
domainLog.deleteDomainLogProfiles(Arrays
                    .asList(new String[] { profileName }));
```

Retrieval of a List of Log Filters

Retrieves a list of domain log filters.

Syntax

```
Collection<DomainLogFilterVO> getDomainLogFilters() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
for (DomainLogFilterVO dlfvo : domainLog.getDomainLogFilters()) {
    System.out.println (dlfvo.getName());
}
```

Creation or Update of a Correlation Log Filter

Persists the domain log filters for later usage.

Syntax

```
void saveDomainLogFilters(Collection<DomainLogFilterVO> filters)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **–**

```
DomainLogFilterVO dlfvo = new
DomainLogFilterVO(DOMAIN_LOG_CATEGORY.ALL);
FilterExpression<DOMAIN_LOG_FILTER> fe = new FilterExpression<
DOMAIN_LOG_FILTER >();
FilterExpression< DOMAIN_LOG_FILTER > fe1 = new FilterExpression<
DOMAIN_LOG_FILTER >();
fe1 = fe.eq(DOMAIN_LOG_FILTER.APPLICATION_CONNECTION_ID,
"emulator1").and(
fe.eq(DOMAIN_LOG_FILTER.DOMAIN,
"default")).or(fe.eq(DOMAIN_LOG_FILTER.PACKAGE, "sap_crm:1.0"));
dlfvo.setFilterExpression(fe1);
domainLog.saveDomainLogFilters(Arrays.asList(new
DomainLogFilterVO[]{dlfvo}));
```

Deletion of a Log Filter

Deletes a log filter.

Syntax

```
void deleteDomainLogFilters(Collection<String> filterNames) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- –

```
domainLog.deleteDomainLogFilters(Arrays
.asList(new String[] { "filter1" }));
```

Retrieval of a List of Log Entries

Retrieves the domain log entries with the given filters, time range, offset and length.

Syntax

```
List<DomainLogEntryVO>
getDomainLogEntry(Collection<DomainLogFilterVO> filters, Date
StartTime, Date EndTime, Long offset, Integer length) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
DomainLogFilterVO dlfvo =
domainLog.getDomainLogFilter("filter1");
List<DomainLogEntryVO> logEntries = domainLog.getDomainLogEntry(
        Arrays.asList(new DomainLogFilterVO[] { dlfvo }), null,
        null, null, null);
for(DomainLogEntryVO dlevo : logEntries){
        for(Map.Entry<String, Object> entry :
dlevo.getEntry().entrySet()){
            System.out.println(entry.getKey() + ":" +
entry.getValue());
        }
}
```

Deletion of Domain Log Entries

Deletes the domain log entries within the specified time range.

Syntax

```
void deleteLog(Date startTime, Date endTime) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

  ```
  domainLog.deleteLog(new Date(0), new Date());
  ```

Retrieval of Log Store Policy

Retrieves the properties of the domain log store policy.

Syntax

```
DomainLogStorePolicyVO getDomainLogStorePolicy() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

  ```
  DomainLogStorePolicyVO dlspvo = supCluster
          .getDomainLogStorePolicy();
  System.out.println(dlspvo.getCurrentDomainLogDataSource());
  System.out.println(dlspvo.getAvailableDomainLogDataSource());
  System.out.println(dlspvo.getDomainLogFlushBatchSize());
  System.out.println(dlspvo.getLazyWriteEnabled());
  System.out.println(dlspvo.getLazyWriteRowThreshold());
  System.out.println(dlspvo.getLazyWriteTimeThreshold());
  System.out.println(dlspvo.getPurgeTimeThreshold());
  ```

Usage

These methods are only accessible to the Platform Administrator.

Update of Log Store Policy

Updates the properties of the domain log store policy.

Syntax

```
void setDomainLogAutoPurgeTimeThreshold(Integer days) throws
SUPAdminException;

void setDomainLogDataSource(String datasource) throws
SUPAdminException;

void setDomainLogFlushBatchSize(Integer rows) throws
SUPAdminException;

void setDomainLogLazyWriteRowThreshold(Integer rowcount) throws
SUPAdminException;

void setDomainLogLazyWriteStatus(Boolean flag) throws
SUPAdminException;

void setDomainLogLazyWriteTimeThreshold(Integer minutes) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

  ```
  supCluster.setDomainLogAutoPurgeTimeThreshold(7);
  supCluster.setDomainLogDataSource("newDomainLogDB");
  supCluster.setDomainLogFlushBatchSize(100);
  supCluster.setDomainLogLazyWriteRowThreshold(200);
  supCluster.setDomainLogLazyWriteStatus(true);
  supCluster.setDomainLogLazyWriteTimeThreshold(100);
  ```

Usage

These methods are only accessible to the Platform Administrator.

Configuring Unwired Servers

Administration of the Unwired Server configuration is provided through the `SUPServerConfiguration` interface.

The Unwired Server configuration consists of the following components, which are all metadata-based configurations, except for Apple Push Notification Service:

- Communication
  - Administration Listener
  - HTTP / HTTPS Listener

  - SSL Security Profile
  - Key Store
  - Trust Store
- Messaging
  - Server
  - Apple Push Notification
- Replication
  - Server
  - Push Notification
  - Push Notification Gateway
  - Pull Notification
- Consolidated DB
- Java Virtual Machine (JVM) startup options
- Apple Push Notification Service

The `SUPServerConfiguration` interface provides different methods for these components. The metadata-based configurations have these characteristics:

- Each of these components is represented by `ServerComponentVO`.
- The properties of `ServerComponentVO` differentiate these components. See *Developer Guide for Unwired Server Management API > Client Metadata*.
- Each instance of `SUPServerConfiguration` is a local object which holds values of all metadata-based configurations. All of its methods perform against those values. The values are refreshed when you call the `commit()`and `refresh()` methods. After you receive an instance of `SUPServerConfiguration`, call the `refresh()` method to populate the values, before calling any other methods.
- Changes made through these methods are cached locally unless the `commit()` method is called. `Commit()` sends all the cached values (whether changed or not) to the Unwired Server. A server restart may be required for some changes to take effect.

ServerComponentVO

The `ServerComponentVO` class has a read-only property that you must initialize at construction time.

The type property specifies the server component type. The server component types are described in *Developer Guide for Unwired Server Management API > Client Metadata > Server Configuration*.

Start Management of Unwired Server Configuration

Starts the management of Unwired Server configuration information.

Syntax

```
public static SUPServerConfiguration
getSUPServerConfiguration(ServerContext serverContext) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Unwired Server configuration –**

  ```
  SUPServerConfiguration supServerConf = SUPObjectFactory
          .getSUPServerConfiguration(serverContext);
  ```

Usage

When an instance of `SUPServerConfiguration` is returned from the SUPObjectFactory, call its method.

Populate Server Configuration

Retrieves the server configuration from the Unwired Server and caches it locally. This method refreshes all metadata-based configuration. The returned `ConfigurationValidationStatus` contains the validation status of the security configuration on the server.

Syntax

```
ConfigurationValidationStatus refresh() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Populate server configuration –**

  ```
  supServerConf.refresh();
  ```

Usage

When you call `SUPServerConfiguration.refresh()`, any data in the local cache is overwritten.

Each call to `commit()` and `refresh()` expire all previous `ServerComponentVOs`, because all the IDs are regenerated.

Commit Local Changes to Unwired Server

Commits local changes to the Unwired Server. The returned `ConfigurationValidationStatus` contains the validation status of the delivered security configuration on the Unwired Server.

Syntax

```
ConfigurationValidationStatus commit() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
ConfigurationValidationStatus cvs = supServerConf.commit();
if(cvs.isValid()){
    //succeed.
}
else{
    //fail.
}
```

Retrieval of Replication Sync Server Configuration

Retrieves the properties of the replication synchronization server configuration.

Syntax

```
ServerComponentVO getReplicationSyncServerConfiguration() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getReplicationSyncServerConfiguration();
System.out.println(scvo.getID());
```

```
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Update of Replication Sync Server Configuration

Updates the properties of the replication synchronization server configuration.

Syntax

```
void updateReplicationSyncServerConfiguration(ServerComponentVO
serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getReplicationSyncServerConfiguration();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.sync.protocol", "http");
properties.put("ml.threadcount", "50");
scvo_new.setProperties(properties);
supServerConf.updateReplicationSyncServerConfiguration(scvo_new);
supServerConf.commit();
```

Retrieval of Replication Push Notification Configuration

Retrieves the properties of the replication notifier configuration of the specified type.

Syntax

```
ServerComponentVO
getReplicationNotifierConfiguration(REPLICATION_NOTIFIER_TYPE
replicationNotifierType) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves the configuration of the replication push notification:

```
supServerConf.refresh();
ServerComponentVO scvo =
```

```
supServerConf.getReplicationNotifierConfiguration
                          (REPLICATION_NOTIFIER_TYPE.PUSH);
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Update of Replication Push Notification Configuration

Updates the configuration of the replication push notification.

Syntax

```
void
updateReplicationNotifierConfiguration(REPLICATION_NOTIFIER_TYPE
replicationNotifierType, ServerComponentVO serverComponent) throws
SUPAdminException;

void enableReplicationPushNotificationGatewayConfiguration(String
serverComponentID, Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** enables and updates the replication push notification configuration:

```
ServerComponentVO scvo = supServerConf
        .getReplicationNotifierConfiguration(REPLICATION_NOTIFIER
_TYPE.PUSH);
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo
        .getType());
Map<String, String> properties = scvo.getProperties();
properties.put("poll_every", "10");
scvo_new.setProperties(properties);
supServerConf.updateReplicationNotifierConfiguration(
        REPLICATION_NOTIFIER_TYPE.PUSH, scvo_new);
supServerConf.enableReplicationNotifierConfiguration(
        REPLICATION_NOTIFIER_TYPE.PUSH, true);
supServerConf.commit();
```

Retrieval of Replication Push Notification Gateway Configuration

Retrieves the configuration of the replication push gateway.

Syntax

```
ServerComponentVO
getReplicationPushNotificationGatewayConfiguration() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getReplicationPushNotificationGatewayConfiguration(
);
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Update of Replication Notification Gateway Configuration

Updates the properties of the replication push gateway configuration.

Syntax

```
void updateReplicationPushNotificationGatewayConfiguration(String
serverComponentID, ServerComponentVO serverComponent) throws
SUPAdminException;

void enableReplicationPushNotificationGatewayConfiguration(String
serverComponentID, Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** enables and updates the properties of the replication push gateway configuration:

```
supServerConf.refresh();
ServerComponentVO scvo = supServerConf
        .getReplicationPushNotificationGatewayConfiguration();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo
        .getType());
Map<String, String> properties = scvo.getProperties();
properties.put("confirm_action", "true");
scvo_new.setProperties(properties);
supServerConf.updateReplicationPushNotificationGatewayConfigurati
on(scvo_new.getID(), scvo_new);
supServerConf.enableReplicationPushNotificationGatewayConfigurati
on(scvo_new.getID(), true);
supServerConf.commit();
```

Retrieval of Messaging Sync Server Configuration

Retrieves the properties of the messaging synchronization configuration from the Unwired Server.

Syntax

```
ServerComponentVO getMessagingSyncServerConfiguration() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getMessagingSyncServerConfiguration();
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Update of Messaging Sync Server Configuration

Updates the properties of the messaging synchronization configuration on the Unwired Server.

Syntax

```
void updateMessagingSyncServerConfiguration(ServerComponentVO
serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** updates the messaging synchronization configuration on the Unwired Server by specifying the ID, Type, and Properties:

```
supServerConf.refresh();
ServerComponentVO scvo = supServerConf
        .getMessagingSyncServerConfiguration();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
```

```
Map<String, String> properties = scvo.getProperties();
properties.put("msg.admin.webservices.port", "5100");
properties.put("msg.http.server.ports", "5001,80");
scvo_new.setProperties(properties);
supServerConf.updateMessagingSyncServerConfiguration(scvo_new);
supServerConf.commit();
```

Retrieval of Consolidated Database Configuration

Retrieves the properties of the consolidated database configuration.

Syntax

```
ServerComponentVO getConsolidatedDatabaseConfiguration() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getConsolidatedDatabaseConfiguration();
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Retrieval of Administration Listener Configuration

Retrieves the configuration of the administration listener.

Syntax

```
ServerComponentVO getAdministrationListenerConfiguration() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
ServerComponentVO scvo =
```

```
supServerConf.getAdministrationListenerConfiguration();
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Update of Administration Listener Configuration

Updates the properties of the administration listener configuration.

Syntax

```
void updateAdministrationListenerConfiguration(String
serverComponentID, ServerComponentVO serverComponent) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getAdministrationListenerConfiguration();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.socket.listener.port", "2000");
properties.put("sup.socket.listener.enabled", "true");
scvo_new.setProperties(properties);
supServerConf.updateAdministrationListenerConfiguration(scvo_new.
getID(), scvo_new);
supServerConf.commit();
```

Retrieval of HTTP Listener Configuration

Retrieves a list of HTTP listener configurations.

Syntax

```
Collection<ServerComponentVO> getHTTPListenerConfigurations() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
for(ServerComponentVO scvo :
supServerConf.getHTTPListenerConfigurations()){
    System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
}
```

Addition of HTTP Listener Configuration

Adds a new HTTP listener configuration.

Syntax

```
void addHTTPListenerConfiguration(ServerComponentVO serverComponent)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Add configuration –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getHTTPListenerConfigurations()
        .iterator().next();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.socket.listener.port", "8000");
properties.put("sup.socket.listener.enabled", "true");
scvo_new.setProperties(properties);
supServerConf.addHTTPListenerConfiguration(scvo_new);
supServerConf.commit();
```

Deletion of HTTP Listener Configuration

Deletes the configuration for an HTTP listener.

Syntax

```
void deleteHTTPListenerConfiguration(String serverComponentID)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getHTTPListenerConfigurations()
        .iterator().next();
supServerConf.deleteHTTPListenerConfiguration(scvo.getID());
supServerConf.commit();
```

Update of HTTP Listener Configuration

Updates the configuration of an HTTP listener.

Syntax

```
void updateHTTPListenerConfiguration(String serverComponentID,
ServerComponentVO serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getHTTPListenerConfigurations()
        .iterator().next();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.socket.listener.port", "8000");
properties.put("sup.socket.listener.enabled", "true");
scvo_new.setProperties(properties);
supServerConf.updateHTTPListenerConfiguration(scvo_new.getID(),
scvo_new);
supServerConf.commit();
```

Retrieval of HTTPS Listener Configuration

Retrieves a list of HTTPS listener configurations.

Syntax

```
Collection<ServerComponentVO> getSecureHTTPListenerConfigurations()
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
for(ServerComponentVO scvo :
supServerConf.getSecureHTTPListenerConfigurations()){
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
}
```

Addition of HTTPS Listener Configuration

Adds a new HTTPS listener configuration.

Syntax

```
void addSecureHTTPListenerConfiguration(ServerComponentVO
serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Add configuration –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getSecureHTTPListenerConfigurations()
        .iterator().next();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.socket.listener.port", "8001");
properties.put("sup.socket.listener.enabled", "true");
scvo_new.setProperties(properties);
supServerConf.addSecureHTTPListenerConfiguration(scvo_new);
supServerConf.commit();
```

Deletion of HTTPS Listener Configuration

Deletes the configuration for a secure HTTP (HTTPS) listener.

Syntax

```
void deleteSecureHTTPListenerConfiguration(String serverComponentID)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getSecureHTTPListenerConfigurations ()
        .iterator().next();
supServerConf.deleteSecureHTTPListenerConfiguration(scvo.getID())
;
supServerConf.commit();
```

Update of HTTPS Listener Configuration

Updates the configuration of an HTTP listener.

Syntax

```
void updateSecureHTTPListenerConfiguration(String serverComponentID,
ServerComponentVO serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getHTTPListenerConfigurations()
        .iterator().next();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.socket.listener.port", "8000");
properties.put("sup.socket.listener.enabled", "true");
```

```
scvo_new.setProperties(properties);
supServerConf.updateHTTPListenerConfiguration(scvo_new.getID(),
scvo_new);
supServerConf.commit();
```

Retrieval of SSL Security Profile Configuration

Retrieves the list of all the SSL security profiles and their properties.

Syntax

```
Collection<ServerComponentVO> getSSLSecurityProfileConfigurations()
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
for(ServerComponentVO scvo :
supServerConf.getSSLSecurityProfileConfigurations()){
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
}
```

Addition of SSL Security Profile Configuration

Adds configuration for an SSL security profile.

Syntax

```
void addSSLSecurityProfileConfiguration(ServerComponentVO
serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Add configuration –** adds configuration for an SSL security profile, including the authentication profile, profile name, and key alias:

```
supServerConf.refresh();
ServerComponentVO scvo = supServerConf
        .getSSLSecurityProfileConfigurations().iterator().next();
```

```
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.security.profile.auth", "domestic");
properties.put("sup.security.profile.name",
        "<SSL security profile name>");
properties.put("sup.security.profile.key.alias",
        "<SSL security key alias>");
scvo_new.setProperties(properties);
supServerConf.addSSLSecurityProfileConfiguration(scvo_new);
supServerConf.commit();
```

Deletion of SSL Security Profile Configuration

Deletes the configuration for an SSL security profile.

Syntax

```
void deleteSSLSecurityProfileConfiguration(String serverComponentID)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion –**

```
supServerConf.refresh();
ServerComponentVO scvo = supServerConf
        .getSSLSecurityProfileConfigurations().iterator().next();
supServerConf.deleteSSLSecurityProfileConfiguration(scvo.getID())
;
supServerConf.commit();
```

Update of SSL Security Profile Configuration

Updates the configuration of an SSL security profile.

Syntax

```
void updateSSLSecurityProfileConfiguration(String serverComponentID,
ServerComponentVO serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** updates the configuration of an SSL security profile, including the authentication profile, profile name, and key alias:

```
supServerConf.refresh();
ServerComponentVO scvo = supServerConf
        .getSSLSecurityProfileConfigurations().iterator().next();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.security.profile.auth", "domestic");
properties.put("sup.security.profile.name",
        "<SSL security profile name>");
properties.put("sup.security.profile.key.alias",
        "<SSL security key alias>");
scvo_new.setProperties(properties);
supServerConf.updateSSLSecurityProfileConfiguration(scvo_new.getI
D(), scvo_new);
supServerConf.commit();
```

Key Store Configuration Retrieval

Retrieves the properties of the key store configuration.

Syntax

```
ServerComponentVO getKeyStoreConfiguration() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getKeyStoreConfiguration();
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Key Store Configuration Update

Updates the configuration of the key store.

Syntax

```
void updateKeyStoreConfiguration(ServerComponentVO serverComponent)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** updates the configuration of the key store, including the key store file path, and key store password:

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getKeyStoreConfiguration();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.sync.sslkeystore", "<key store file path>");
properties.put("sup.sync.sslkeystore_password", "<key store
password>");
scvo_new.setProperties(properties);
supServerConf.updateKeyStoreConfiguration(scvo_new);
supServerConf.commit();
```

Trust Store Configuration Retrieval

Retrieves the properties of the trust store configuration.

Syntax

```
ServerComponentVO getTrustStoreConfiguration() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getTrustStoreConfiguration();
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Trust Store Configuration Update

Updates the configuration of the trust store.

Syntax

```
void updateTrustStoreConfiguration(ServerComponentVO
serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** updates the configuration of the trust store, including the trust store file path and trust store password:

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getTrustStoreConfiguration();
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("sup.sync.ssltruststore", "<trust store file
path>");
properties.put("sup.sync.ssltruststore_password", "<trust store
password>");
scvo_new.setProperties(properties);
supServerConf.updateTrustStoreConfiguration(scvo_new);
supServerConf.commit();
```

Retrieval of Apple Push Notification Configurations

Retrieves Apple Push Notification configurations.

Syntax

```
List<APNSApplicationSettingsVO>
getApplePushNotificationConfigurations(boolean getPendingConfig)
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval: getPendingConfig is true –** retrieves Apple Push Notification application settings that are applied to the Unwired Server the next time the Unwired Server starts:

```
// List Apple push configuration
List<APNSAppSettingsVO> list =
supServerConf.getApplePushNotificationConfigurations(true);
```

- **Retrieval: getPendingConfig is false –** retrieves current Apple Push Notification application settings:

```
// List Apple push configuration
List<APNSAppSettingsVO> list =
supServerConf.getApplePushNotificationConfigurations(false);
```

Addition of an Apple Push Notification Configuration

Adds a configuration for Apple Push Notification.

Syntax

```
void
addApplePushNotificationConfiguration(APNSApplicationSettingsVO
settings, byte[] p12Certificate, boolean overwrite, boolean restart)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful (for example, if a certificate of the same name exists and `overwrite` is false), returns `SUPAdminException`.

Examples

- **Add configuration –**

```
// Add Apple push configuration
APNSAppSettingsVO settings = buildAPNSSettings();
byte[] certificate = getCertificate();
supServerConf.addApplePushNotificationConfiguration(settings,
certificate, false, false);
```

Deletion of an Apple Push Notification Configuration

Deletes an Apple Push Notification configuration.

Syntax

```
Boolean deleteApplePushNotificationConfiguration(String
apnsConfigName, boolean restart) throws SUPAdminException;
```

Returns

If successful, returns true if the specified APNS configuration has been removed, or false if the specified APNS configuration does not exist. If unsuccessful, returns `SUPAdminException`.

Examples

- **Removal –**

```
// Delete Apple push configuration
supServerConf.deleteApplePushNotificationConfiguration("smithj_AP
NS_configuration1", false);
```

Update of an Apple Push Notification Configuration

Updates an Apple Push Notification configuration.

Syntax

```
void
updateApplePushNotificationConfiguration(APNSApplicationSettingsVO
settings, byte[] p12Certificate, boolean overwrite, boolean restart)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** updates an Apple Push Notification configuration including the feedback gateway and the Apple Push Notification settings:

```
// Update Apple push configuration
APNSAppSettingsVO settings = buildAPNSSettings();
settings.setFeedbackGateway("testfeedback.push2.example.com ");
byte[] certificate = getCertificate();
supServerConf.updateApplePushNotificationConfiguration(settings,
certificate, true, false);
```

Retrieval of Certificate Names

Retrieves a list of file names for the .p12 certificates on the Unwired Server.

Syntax

```
List<String> getApplePushNotificationCertificateNames() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
// List APNS certificate names
List<String> list =
supServerConf.getApplePushNotificationCertificateNames();
```

Set Apple Notification Values

Constructs a value object, `APNSAppSettingsVO`, which sets values for Apple Push Notification Service settings used for iPhone push notifications.

Syntax

```
public java.lang.String getCertificateFileName()
public void setCertificateFileName(java.lang.String value)
public java.lang.String getCertificatePassword()
public void setCertificatePassword(java.lang.String value)
public java.lang.String getPushGateway()
public void setPushGateway(java.lang.String value)
public int getPushGatewayPort()
public void setPushGatewayPort(int value)
public int getNumberOfChannels()
public void setNumberOfChannels(int value)
public java.lang.String getFeedbackGateway()
public void setFeedbackGateway(java.lang.String value)
public int getFeedbackGatewayPort()
public void setFeedbackGatewayPort(int value)
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
// construct an APNSAppSettingsVO
private APNSAppSettingsVO buildAPNSSettings() {
    APNSAppSettingsVO settings = new APNSAppSettingsVO();
    settings.setCertificateFileName("C:/
PushDevCertificate_smithj.p12");
    settings.setCertificatePassword("iM0;APNS");
    settings.setFeedbackGateway("testfeedback.push.example.com");
    settings.setFeedbackGatewayPort(123);
    settings.setName("smithj_APNS_configuration1");
    settings.setNumberOfChannels(3);
```

```
    settings.setPushGateway("testgateway.push.example.com ");
    settings.setPushGatewayPort(456);
    return settings;
}
```

Retrieval of Replication Pull Notification Configuration

Retrieves the configuration of the replication pull notification.

Syntax

```
ServerComponentVO
getReplicationNotifierConfiguration(REPLICATION_NOTIFIER_TYPE
replicationNotiferType) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –** retrieves the configuration of the replication pull notification, including the ID, type, and properties:

```
supServerConf.refresh();
ServerComponentVO scvo = supServerConf
        .getReplicationNotifierConfiguration(REPLICATION_NOTIFIER
_TYPE.PULL);
System.out.println(scvo.getID());
System.out.println(scvo.getType());
System.out.println(scvo.getProperties());
```

Update of Replication Pull Notification Configuration

Updates the configuration of the replication pull notification.

Syntax

```
void
updateReplicationNotifierConfiguration(REPLICATION_NOTIFIER_TYPE
replicationNotifierType, ServerComponentVO serverComponent) throws
SUPAdminException;

void
enableReplicationNotifierConfiguration(REPLICATION_NOTIFIER_TYPE
replicationNotiferType, Boolean flag) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –** updates the configuration of the replication pull notification, including the ID, type, and properties:

```
supServerConf.refresh();
ServerComponentVO scvo =
supServerConf.getReplicationNotifierConfiguration(REPLICATION_NOT
IFIER_TYPE.PULL);
ServerComponentVO scvo_new = new ServerComponentVO(scvo.getID(),
scvo.getType());
Map<String, String> properties = scvo.getProperties();
properties.put("poll_every", "10");
scvo_new.setProperties(properties);
supServerConf.updateReplicationNotifierConfiguration(REPLICATION_
NOTIFIER_TYPE.PULL, scvo_new);
supServerConf.enableReplicationNotifierConfiguration(REPLICATION_
NOTIFIER_TYPE.PULL, true);
supServerConf.commit();
```

Update Server Configuration for Relay Server

Updates the server configuration for Relay Server.

Syntax

```
void updateServerConfigurationForRelayServer(ServerComponentVO
serverComponent) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
// get
supServerConf.refresh();
ServerComponentVO componentVO = supServerConf
        .getServerConfigurationForRelayServer();
System.out.println(componentVO.getProperties());
// update
componentVO.getProperties().put("relayserver.trusted_certs",
        "Repository/Security/myRelayServerTrustedCert.cert");
supServerConf.updateServerConfigurationForRelayServer(componentVO
);
supServerConf.commit();
```

Retrieval of Relay Server Outbound Enablers

Retrieves the Relay Server Outbound Enablers for the Unwired Server.

Syntax

```
List<OutboundEnablerVO> getOutboundEnablers() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
// Get Outbound Enablers of the target Unwired Server.
List<OutboundEnablerVO> outboundEnablers = supServerConf
        .getOutboundEnablers();
for (OutboundEnablerVO enabler : outboundEnablers) {
    // Print out the Outbound Enabler Info
    System.out.println("=======Begin Outbound Enabler
Info======");
    System.out.println("Unwired Server Host: "
            + enabler.getUnwiredServerHost());
    System.out.println("Unwired Server Port: "
            + enabler.getUnwiredServerPort());
    System.out.println("Unwired Server Name: "
            + enabler.getUnwiredServerName());
    System.out
            .println("Outbound Enabler connect relay server via
HTTPS port: "
                    + enabler.isUseSecureRelayServerPort());
    ServerNodeVO serverNode = enabler.getServerNode();
    System.out.println("The server node name: " +
serverNode.getName());
    FarmVO farm = serverNode.getFarm();
    System.out.println("The farm name: " + farm.getName());
    RelayServerVO relayServer = farm.getRelayServer();
    System.out.println("The relay server host: "
            + relayServer.getHost());
    System.out.println("The relay server HTTP prot: "
            + relayServer.getPort());
    System.out.println("The relay server HTTPS port: "
            + relayServer.getSecurePort());
    System.out.println("=======End Outbound Enabler Info======");
}
```

Configuring Security Configurations

The Sybase Unwired Platform security configuration is a metadata-based configuration that includes several components.

- Authentication provider

- Authorization provider
- Audit provider

Each of these components is a security provider, and is represented by `SecurityProviderVO`. The properties of `SecurityProviderVO` differentiate the components. See *Developer Guide for Unwired Server Management API > Client Metadata*.

Manage the Sybase Unwired Platform security configuration using the `SUPSecurityConfiguration` interface. This interface provides different methods for the components. The changes made through these methods are cached locally unless the `commit()` method is called to send the cached configuration of all the components to the Unwired Server.

Start Security Configuration Management

Starts the management of an Unwired Server security configuration.

Syntax

```
public static SUPSecurityConfiguration
getSUPSecurityConfiguration(SecurityContext securityContext) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Security configuration –**

```
//Retrieve a list of security configuration names currently
defined
Collection<String> securityConfigurations=
supCluster.getSecurityConfigurations();

//Start administration on one of the security configurations
securityContext = serverContext.getSecurityContext("<security
configuration name>");
SUPSecurityConfiguration supSecConf =
SUPObjectFactory.getSUPSecurityConfiguration(securityContext);
```

Usage

When an instance of `SUPSecurityConfiguration` is returned from the `SUPObjectFactory`, call its method.

SecurityProviderVO

The `ServerProviderVO` class has a read-only property that you must initialize at construction time.

The type property specifies the provider type, as described in *Developer Guide for Unwired Server Management API > Client Metadata > Security Configuration*.

Populate Security Configuration

Populates an Unwired Server security configuration with the currently effective configuration. The returned `ConfigurationValidationStatus` contains the validation status of the security configuration on the Unwired Server.

Syntax

```
ConfigurationValidationStatus refresh() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Populate security configuration –**
  ```
  supSecConf.refresh();
  ```

Usage

Each call to `commit()` and `refresh()` expires all the previous `ServerProviderVO`, because all the IDs are regenerated.

`supSecConf.refresh()` retrieves from the Unwired Server the current configuration, which does not include any committed changes that are pending a server restart, and caches it locally.

Commit Local Changes to the Unwired Server

Commits local changes to the Unwired Server. The returned `ConfigurationValidationStatus` contains the validation status of the security configuration on the Unwired Server.

Syntax

```
ConfigurationValidationStatus commit() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Commit local changes –**

```
ConfigurationValidationStatus cvs = supServerConf.commit();
if(cvs.isValid()){
    //succeed.
}
else{
    //fail.
}
```

Active Security Providers

Active security providers are those that are currently effective on the Unwired Server. Each active security provider has a location in the respective active security provider stack. These locations are reflected in the sequence when iterating through the returned collection. You can retrieve, update, add, or delete active security providers.

Retrieval of Active Security Providers

Retrieves the active security providers.

Syntax

```
public SecurityProviderVO getActiveAuditProvider(String
auditProviderID) throws SUPAdminException;

public SecurityProviderVO getActiveAuthenticationProvider(String
authenticationProviderID) throws SUPAdminException;

public SecurityProviderVO getActiveAuthorizationProvider(String
authorizationProviderID) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval –**

```
supSecConf.refresh();

Collection<SecurityProviderVO> spvos_audit =
supSecConf.getActiveAuditProviders();
```

```
SecurityProviderVO spvo_audit =
supSecConf.getActiveAuditProvider("<security provider id>");

Collection<SecurityProviderVO> spvos_authentication =
supSecConf.getActiveAuthenticationProviders();
SecurityProviderVO spvo_authentication =
supSecConf.getActiveAuthenticationProvider("<security provider
id>");

Collection<SecurityProviderVO> spvos_authorization =
supSecConf.getActiveAuthorizationProviders();
SecurityProviderVO spvo_authorization =
supSecConf.getActiveAuthorizationProvider("<security provider
id>");
```

Update of Active Security Providers

Updates the active security providers, including the active attribution provider, audit provider, authentication provider, or authorization provider.

Syntax

```
public void updateActiveAuditProvider(String auditProviderID,
SecurityProviderVO securityProvider) throws SUPAdminException;

public void updateActiveAuthenticationProvider(String
authenticationProviderID, SecurityProviderVO securityProvider)
throws SUPAdminException;

public void updateActiveAuthorizationProvider(String
authorizationProviderID, SecurityProviderVO securityProvider) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Update –**

```
supSecConf.refresh();
SecurityProviderVO spvo_audit =
supSecConf.getActiveAuditProviders()
        .iterator().next();
SecurityProviderVO spvo_authentication = supSecConf
        .getActiveAuthenticationProviders().iterator().next();
SecurityProviderVO spvo_authorization = supSecConf
        .getActiveAuthorizationProviders().iterator().next();
supSecConf.updateActiveAuditProvider("<security provider id>",
spvo_audit);
supSecConf.updateActiveAuthenticationProvider("<security provider
id>", spvo_authentication);
supSecConf.updateActiveAuthorizationProvider("<security provider
```

```
id>",    spvo_authorization);
supSecConf.commit();
```

Addition of an Active Authentication Provider

Adds an active authentication provider.

Syntax

```
public void addActiveAuthenticationProvider(SecurityProviderVO
securityProvider) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Add active authentication provider –**

```
supSecConf.refresh();

SecurityProviderVO spvo = new SecurityProviderVO(
       "com.sybase.security.ldap.LDAPLoginModule");
Map<String, String> properties = new HashMap<String, String>();
spvo.setProperties(properties);
//Mandatory properties.
properties.put("implementationClass",
       "com.sybase.security.ldap.LDAPLoginModule");
properties.put("providerType", "LoginModule");
properties.put("ProviderURL", "ldap://localhost:389");
properties.put("controlFlag", "optional");
//Optional properties.
properties.put("ServerType", "sunone5");

spvo.setProperties(properties);
supSecConf.addActiveAuthenticationProvider(spvo);
supSecConf.commit();
```

Addition of an Active Authorization Provider

Adds an active authorization provider.

Syntax

```
public void addActiveAuthorizationProvider(SecurityProviderVO
securityProvider) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Add active authorization provider –**

```
supSecConf.refresh();

SecurityProviderVO spvo = new SecurityProviderVO(
        "com.sybase.security.ldap.LDAPAuthorizer");
Map<String, String> properties = new HashMap<String, String>();
spvo.setProperties(properties);
// Mandatory properties.
properties.put("implementationClass",
        "com.sybase.security.ldap.LDAPAuthorizer");
properties.put("providerType", "Authorizer");
// Optional properties.
properties.put("ProviderURL", "ldap://localhost:389");
properties.put("ServerType", "sunone5");

spvo.setProperties(properties);
supSecConf.addActiveAuthorizationProvider(spvo);
supSecConf.commit();
```

Addition of an Active Audit Provider

Adds an active audit provider.

Syntax

```
public void addActiveAuditProvider(SecurityProviderVO
securityProvider) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Add active audit provider –**

```
supSecConf.refresh();

SecurityProviderVO spvo = new SecurityProviderVO("auditor");

SecurityProviderVO spvo_dest = new SecurityProviderVO(
        "auditDestination");
SecurityProviderVO spvo_filter = new
SecurityProviderVO("auditFilter");
SecurityProviderVO spvo_formatter = new SecurityProviderVO(
        "auditFormatter");

Map<String, String> properties_dest = new HashMap<String,
String>();
Map<String, String> properties_filter = new HashMap<String,
String>();
Map<String, String> properties_formatter = new HashMap<String,
```

```
String>();

properties_dest.put("controlFlag", "optional");
properties_dest.put("implementationClass", "");
properties_dest.put("providerType", "AuditDestination");

properties_filter.put("implementationClass", "");
properties_filter.put("providerType", "AuditFilter");

properties_formatter.put("implementationClass", "");
properties_formatter.put("providerType", "AuditFormatter");

spvo_dest.setProperties(properties_dest);
spvo_filter.setProperties(properties_filter);
spvo_formatter.setProperties(properties_formatter);

spvo.setChildren(Arrays.asList(new SecurityProviderVO[]
{ spvo_dest, spvo_filter, spvo_formatter }));

supSecConf.addActiveAuditProvider(spvo);
supSecConf.commit();
```

Deletion of an Active Security Provider

Deletes an active security provider.

Syntax

```
public void deleteActiveAuditProvider(String auditProviderID) throws
SUPAdminException;

public void deleteActiveAuthenticationProvider(String
authenticationProviderID) throws SUPAdminException;

public void deleteActiveAuthorizationProvider(String
authorizationProviderID) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Delete –**

```
supSecConf.refresh();

supSecConf.deleteActiveAuditProvider("<security provider id>");
supSecConf.deleteActiveAuthenticationProvider("<security provider
id>");
supSecConf.deleteActiveAuthorizationProvider("<security provider
id>");

supSecConf.commit();
```

Security Configuration Validation

Delivers modified Sybase Unwired Platform security configuration to the Unwired Server for validation. The current Unwired Server security configuration is not affected.

Syntax

```
ConfigurationValidationStatus validate() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Validation –**

```
ConfigurationValidationStatus cvs = supSecConf.validate();
if(cvs.isValid()){
    //valid.
}
else{
    //invalid.
}
```

Adjustment of the Sequence of Active Security Providers

Security provider instances are grouped together by their provider types (attribution provider, audit provider, authentication provider, and authorization provider) and ordered in a sequence.

The following methods adjust the sequence of security providers in each group.

Syntax

```
public void moveDownActiveAuditProvider(String auditProviderID)
throws SUPAdminException;

public void moveDownActiveAuthenticationProvider(String
authenticationProviderID) throws SUPAdminException;

public void moveDownActiveAuthorizationProvider(String
authorizationProviderID) throws SUPAdminException;



public void moveUpActiveAuditProvider(String auditProviderID) throws
SUPAdminException;

public void moveUpActiveAuthenticationProvider(String
authenticationProviderID) throws SUPAdminException;
```

```
public void moveUpActiveAuthorizationProvider(String
authorizationProviderID) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Adjust sequence –**

```
supSecConf.refresh();

supSecConf.moveDownActiveAuditProvider("<security provider id>");
supSecConf.moveDownActiveAuthenticationProvider("<security
provider id>");
supSecConf.moveDownActiveAuthorizationProvider("<security
provider id>");
supSecConf.commit();

supSecConf.moveUpActiveAuditProvider("<security provider id>");
supSecConf.moveUpActiveAuthenticationProvider("<security provider
id>");
supSecConf.moveUpActiveAuthorizationProvider("<security provider
id>");
supSecConf.commit();
```

Retrieval of Installed Security Providers

Retrieves a list of the security providers installed in the Unwired Server.

Syntax

```
public Collection<String> getInstalledAuditDestinationProviders()
throws SUPAdminException;

public Collection<String> getInstalledAuditFilterProviders() throws
SUPAdminException;

public Collection<String> getInstalledAuditFormatterProviders()
throws SUPAdminException;

public Collection<String> getInstalledAuthenticationProviders()
throws SUPAdminException;

public Collection<String> getInstalledAuthorizationProviders()
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval of installed security providers –**

```
supSecConf.refresh();

Collection<String> spvos_audit_dest = supSecConf
        .getInstalledAuditDestinationProviders();
Collection<String> spvos_audit_filter = supSecConf
        .getInstalledAuditFilterProviders();
Collection<String> spvos_audit_formatter = supSecConf
        .getInstalledAuditFormatterProviders();
Collection<String> spvos_authentication = supSecConf
        .getInstalledAuthenticationProviders();
Collection<String> spvos_authorization = supSecConf
        .getInstalledAuthorizationProviders();
```

Managing Mobile Workflows

Mobile workflow packages, typically created through the Mobile Workflow Application Designer, allow a developer to design mobile workflow screens that can call on the create, update, and delete operations, as well as object queries, of a mobile business object.

You can manage mobile workflow packages through the `SUPWorkflow` interface. Operations you can perform with this interface include:

- Starting administration of mobile workflow packages
- Package management and installation: listing packages, installing packages, and deleting packages
- Retrieving matching rules, context variables, error lists, and queue items
- Updating properties, matching rules, and context variables
- Managing mobile workflow device assignment
- Managing e-mail settings

Start Management of Mobile Workflow Packages

Starts the management of mobile workflow packages.

Syntax

```
public static SUPMobileWorkflow getSUPMobileWorkflow(ClusterContext
clusterContext) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Start mobile workflow package management –**

```
...
private SUPMobileWorkflow workflow;
...
ServerContext serverContext = new ServerContext("wangf-dell",
2000, "supAdmin", "s3pAdmin", false);
clusterContext = serverContext.getClusterContext("wangf's
cluster");
workflow = SUPObjectFactory.getSUPMobileWorkflow(clusterContext);
```

Usage

To manage Unwired Server mobile workflow packages, you must create an instance of `ServerContext` with the correct information, and pass it to `SUPObjectFactory.getSUPMobileWorkflow()`. When an instance of `SUPMobileWorkflow` is returned, you can call its method as a typical Java method call.

Mobile Workflow Package Retrieval

Retrieves a list of mobile workflow packages.

Syntax

```
List<MobileWorkflowVO> getMobileWorkflowList() throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow package retrieval –**

```
// List workflows
List<WorkflowVO> workflows = workflow.getMobileWorkflowList();
```

Installation of a Mobile Workflow Package

Installs a mobile workflow package.

Syntax

```
MobileWorkflowIDVO installMobileWorkflow(byte[]
zippedWorkflowPackage) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow package installation –** This code fragment installs a mobile workflow package named `ActivitiesPackage.zip`, and returns the package name once it is successfully installed:

```
// Install workflow
byte[] workflowBytes= getWorkflowBytes();
MobileWorkflowIDVO workflowID = workflow
                .installMobileWorkflow(zippedWorkflowPackage);

private byte[] getWorkflowBytes() throws URISyntaxException,
IOException {
    String ZIP_NAME = "C:/ActivitiesPackage.zip";
    File zipFile = new File(ZIP_NAME);
    byte[] zippedWorkflowPackage = new byte[(int)
zipFile.length()];
    DataInputStream inputStream = new DataInputStream(new
FileInputStream(
            zipFile));
    inputStream.readFully(zippedWorkflowPackage);
    return zippedWorkflowPackage;
}
```

Deletion of a Mobile Workflow Package

Deletes the specified mobile workflow package.

Syntax

```
void deleteMobileWorkflow(MobileWorkflowIDVO workflowID) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow package deletion –** This code fragment deletes a mobile workflow package with the specified workflow ID:

```
// delete workflow
workflow.deleteMobileWorkflow(workflowID);
```

Retrieval of Matching Rules

Retrieves matching rules for the specified mobile workflow package.

Matching rules are used by the email listener to identify e-mails that match the rules specified by the administrator. When an e-mail message matches the rule, Unwired Server sends the e-mail message as a workflow to the device that matches the rule.

Syntax

```
MobileWorkflowMatchingRulesVO
getMobileWorkflowMatchingRule(MobileWorkflowIDVO workflowID) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow matching rules –**

  ```
  // Get workflow Matching rule
  MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
  workflowID.setVersion(1);
  workflowID.setWID(6);
  MobileWorkflowMatchingRulesVO vo =
  workflow.getMobileWorkflowMatchingRule(workflowID);
  ```

Retrieval of Context Variables

Retrieves context variables for the specified mobile workflow package.

Context variables customize how data is loaded into the Unwired Server cache. You can use context variables to create a smaller, more focused data set that may yield improved performance.

Syntax

```
List<MobileWorkflowContextVariableVO>
getMobileWorkflowContextVariables(MobileWorkflowIDVO workflowID)
throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow context variables –** This code fragment retrieves context variables for the specified mobile workflow package:

```
// Get workflow context variables
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setVersion(1);
workflowID.setWID(6);
List<WorkflowContextVariableVO> list = workflow
        .getMobileWorkflowContextVariables(workflowID);
```

Retrieval of an Error List

Retrieves an error list for the specified mobile workflow package for the specified time period, and paginates the results.

Syntax

```
PaginationResult<MobileWorkflowErrorVO>
getMobileWorkflowErrorList(int startIndex, int maxRecordsToReturn,
MobileWorkflowIDVO id, String userName, Calendar startDate, Calendar
endDate, String orderByField, boolean bAscending) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow error list –** retrieves an error list for the mobile workflow package starting from the date September 30, 2009:

```
// Get workflow error list
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setVersion(1);
workflowID.setWID(7);
Calendar startDate = Calendar.getInstance();
startDate.set(2009, 9, 30);
Calendar endDate = Calendar.getInstance();
PaginationResult<WorkflowErrorVO> list = workflow
                .getMobileWorkflowErrorList(0, 1, workflowID,
"TEST4", startDate,
                        endDate, null, true);
```

Retrieval and Management of Queue Items

Retrieves a list of queue items for the specified Mobile Workflow package, and deletes the specified queue items.

Syntax

```
PaginationResult<MobileWorkflowQueueItemVO>
getMobileWorkflowQueueItems(int startIndex, int maxRecordsToReturn,
MobileWorkflowIDVO id, List<Integer> deviceIDs, List<String>
userNames, String orderByField, boolean ascending) throws
SUPAdminException;

void deleteMobileWorkflowQueueItem(Integer queueItemID, Boolean
forTransformQueue) throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow queue items –**

  ```
  // Get workflow queue items
  MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
  workflowID.setVersion(1);
  workflowID.setWID(1);
  List<Integer> deviceIds = new ArrayList<Integer>();
  deviceIds.add(4);
  PaginationResult<MobileWorkflowQueueItemVO> list = workflow
          .fetchWorkflowQueueItems(0, 2, workflowID, null, null,
  null, false);

  //Delete MobileWorkflow queue items.
  workflow.deleteMobileWorkflowQueueItem(1, true);
  ```

Update of Properties

Updates the properties for the specified Mobile Workflow package.

Syntax

```
void updateMobileWorkflowDisplayName(MobileWorkflowIDVO workflowID,
String displayName) throws SUPAdminException;

void updateMobileWorkflowIconIndex(MobileWorkflowIDVO workflowID,
int iconIndex) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow properties –** updates the display name and icon index for the specified Mobile Workflow package:

```
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setVersion(1);
workflowID.setWID(6);

// Update workflow display name
workflow.updateMobileWorkflowDisplayName(workflowID, ":)");

// Update workflow icon index
workflow.updateMobileWorkflowIconIndex(workflowID, 100);
```

Update of Matching Rules

Updates a matching rule for the specified Mobile Workflow package.

Syntax

```
void updateMobileWorkflowMatchingRule(MobileWorkflowIDVO workflowID,
MobileWorkflowMatchingRulesVO matchRule) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow matching rules –**

```
// Update workflow matching rule
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setVersion(1);
workflowID.setWID(6);
MobileWorkflowMatchingRulesVO matchRule = workflow
        .getWorkflowMatchingRule(workflowID);
matchRule.setBODYExpressionType(MobileWorkflowMatchingRulesVO.EXP
RESSION_TYPE_REGULAREXPRESSION);
matchRule.setBODYExpression(".*wang.*");
workflow.updateMobileWorkflowMatchingRule(workflowID, matchRule);
```

Update of Context Variables

Updates context variables for the specified Mobile Workflow package.

Syntax

```
void updateMobileWorkflowContextVariables(MobileWorkflowIDVO
workflowID, List<MobileWorkflowContextVariableVO> contextVariables)
throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow context variables –** updates context variables for an existing mobile workflow package with workflow ID 2:

```
// Update MobileWorkflow context variables
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
// ID 2 version 1 is a existing Mobile Workflow on the server
workflowID.setVersion(1);
workflowID.setWID(2);
List<MobileWorkflowContextVariableVO> contextVariables = workflow
        .getMobileWorkflowContextVariables(workflowID);
contextVariables.get(0).setValue("string value updated");
workflow.updateMobileWorkflowContextVariables(workflowID,contextV
ariables);
```

Usage

For mobile workflow packages that do not support certificate-based authentication, use the following context variables to specify credentials:

- SupUser
- SupPassword

For mobile workflow packages that support certificate-based authentication, use the above variables and the following additional context variables:

- SupCertificateIssuer
- SupCertificateSubject
- SupCertificateNotAfter
- SupCertificateNotBefore

Note: In this case, all the context variables are read-only.

Retrieval of Mobile Workflow Device Status

Retrieves mobile workflow status for a device from the value object `DeviceMobileWorkflowStatusVO`.

Syntax

```
List<DeviceMobileWorkflowStatusVO>
getDeviceMobileWorkflowStatus(MobileWorkflowIDVO workflowID) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow device assignments –**

```
// get MobileWorkflow assignment info
List<DeviceMobileWorkflowStatusVO> list = workflow
        .getDeviceMobileWorkflowStatus(workflowID);
```

Assignment of a Workflow Package

Defines a mobile workflow package and devices, and assigns the package to the devices.

Syntax

```
void assignMobileWorkflowToDevices(MobileWorkflowIDVO workflowID,
List<Integer> deviceIDs) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Package assignment –**

```
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setVersion(1);
workflowID.setWID(2);
List<Integer> deviceIDs = new ArrayList<Integer>();
deviceIDs.add(64);
// assign MobileWorkflow to devices
workflow.assignMobileWorkflowToDevices(workflowID, deviceIDs);
```

Unassignment of a Workflow Package

Unassigns a Mobile Workflow package from devices.

Syntax

```
void unassignMobileWorkflowFromDevices(MobileWorkflowIDVO
workflowID, List<Integer> deviceIDs) throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Package unassignment –**

```
// unassign MobileWorkflow to devices
workflow.unassignMobileWorkflowFromDevices(workflowID,
deviceIDs);
```

Retrieval of Device Workflow Assignments

Retrieves all mobile workflow packages that are assigned to the specified device.

Syntax

```
List<MobileWorkflowAssignmentVO>
getDeviceWorkflowAssignments(Integer deviceLogicalID) throws
SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieve mobile workflow device assignments –**

```
// get all MobileWorkflows that assign to the device. Where 3 is a
// existing device ID.
List<MobileWorkflowAssignmentVO> assignments = workflow
        .getDeviceWorkflowAssignments(3);
```

E-mail Settings Configuration

Updates or retrieves the e-mail settings for a mobile workflow package.

E-mail settings allow the administrator to configure a listener to scan all incoming e-mail messages delivered to an inbox that the administrator indicates during configuration.

Syntax

```
Boolean testEmailConnection(String configXml) throws
SUPAdminException;

void configureEmail(String configurationXML) throws
SUPAdminException;

void enableEmail(boolean enable) throws SUPAdminException;

String getEmailConfiguration() throws SUPAdminException;

Boolean isEmailEnabled() throws SUPAdminException;
```

Returns

If successful, returns an object of the specified type (can be null). If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow e-mail settings –**

```
String configXmlString = readEmailConfig();

// Test Email Multicast connection
Boolean test = workflow.testEmailConnection(config);

// Config Email Multicast
workflow.configureEmail(config);

// Enable Email Multicast
workflow.enableEmail(true);

// Get Email Multicast configuration
String config = workflow.getEmailConfiguration();

// Check if Email Multicast enabled
boolean enable = workflow.isEmailEnabled();

// Read Email Multicast config XML string from file
private String readEmailConfig() throws IOException {
StringBuffer sb = new StringBuffer();
    InputStream in = getClass().getResourceAsStream(
       "/com/sybase/sup/example/email/EmailMulticastConfig.xml");
    BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
    String line;
    while ((line = reader.readLine()) != null) {
        sb.append(line);
        System.out.println(line);
    }
    reader.close();
    return sb.toString();
}
```

Unblock Mobile Workflow Queue

Unblocks the mobile workflow queue for the selected workflows and devices.

Syntax

```
void unblockWorkflowQueueForDevices(MobileWorkflowIDVO workflowID,
List<Integer> deviceIDs, Boolean forTransformQueue) throws
SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Mobile workflow queue –**

```
// prepare mobile workflow ID
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setVersion(100);
workflowID.setWID(2);
// prepare device ids
List<Integer> deviceIDs = new ArrayList<Integer>();
deviceIDs.add(1);
deviceIDs.add(2);
// Unblock mobile workflow queue for devices
workflow.unblockWorkflowQueueForDevices(workflowID, deviceIDs,
true);
```

Replace Mobile Workflow Certificate

Replaces the certificate for a mobile workflow package.

Syntax

```
void replaceMobileWorkflowCertificate(workflowID,
    baos.toByteArray(), "password");
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Replace certificate –**

```
InputStream is = workflowRL.getResourceAsStream("sybase101.p12");
ByteArrayOutputStream baos = new ByteArrayOutputStream();
byte[] buf = new byte[512];
int count;
while ((count = is.read(buf)) != -1) {
    baos.write(buf, 0, count);
}
is.close();
baos.flush();
baos.close();
MobileWorkflowIDVO workflowID = new MobileWorkflowIDVO();
workflowID.setWID(4);
workflowID.setVersion(1);

workflow.replaceMobileWorkflowCertificate(workflowID,
    baos.toByteArray(), "password");
```

Client Application Shutdown

Releases resources currently held by the API. This method only needs to be called on the termination of the client application.

Syntax

```
public static void shutdown() throws SUPAdminException;
```

Returns

If successful, returns silently. If unsuccessful, returns `SUPAdminException`.

Examples

- **Shutdown –**
  ```
  SUPObjectFactory.shutdown();
  ```

Client Metadata

Use metadata to add values the administrator can use to configure Unwired Platform properties.

See also

- *Administration Interfaces* on page 4
- *Metadata* on page 6

Security Configuration

The security configuration for Sybase Unwired Platform consists of the several types of security provider.

- Authentication provider
- Authorization provider
- Audit provider

Each of these provider types can have multiple instances in the security configuration. For example, a security configuration could have two audit providers, four authentication providers, and five authorization providers. Each security provider instance has a unique ID.

Security provider instances are grouped together by type; the instance stack sequence in each group can be adjusted.

Audit Provider

An auditor consists of one destination, one filter, and one formatter:

- The only supported value for destination is `com.sybase.security.core.FileAuditDestination`.
- The only supported value for the filter is `com.sybase.security.core.DefaultAuditFilter`.
- The only supported value for the formatter is `com.sybase.security.core.XmlAuditFormatter`.

com.sybase.security.core.FileAuditDestination

The com.sybase.security.core.FileAuditDestination package contains the following configurable properties:

Table 1. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 2. implementationClass

Datatype	String
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 3. providerType

Datatype	String
Default	AuditDestination
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 4. auditFile

Datatype	String
Default	../logs/audit.log

Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 5. compressionThreshold

Datatype	int
Required?	No
Requires server restart?	No
Read-only?	No

Table 6. deleteThreshold

Datatype	int
Required?	No
Requires server restart?	No
Read-only?	No

Table 7. encoding

Datatype	String
Default	utf-8
Required?	No
Requires server restart?	No
Read-only?	No

Table 8. errorThreshold

Datatype	int
Required?	No
Requires server restart?	No
Read-only?	No

Table 9. logSize

Datatype	long

Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.core.DefaultAuditFilter

The com.sybase.security.core.DefaultAuditFilter package contains the following configurable properties:

Table 10. implementationClass

Datatype	String
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 11. providerType

Datatype	String
Default	AuditFilter
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 12. caseSensitiveFiltering

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 13. filter

Datatype	String

Default	(ResourceClass=core.subject, Action=authorization.role) (ResourceClass=core.subject, Action=authorization.resource) (ResourceClass=core.subject, Action=authentication) (ResourceClass=core.subject, Action=logout) (ResourceClass=core.profile) (ResourceClass=providers.*) (ResourceClass=clients.*)
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.core.XmlAuditFormatter

The com.sybase.security.core.XmlAuditFormatter package contains the following configurable properties:

Table 14. implementationClass

Datatype	String
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 15. providerType

Datatype	String
Default	AuditFormatter
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Authentication Provider

Supported authenticators.

- com.sybase.security.core.NoSecLoginModule
- com.sybase.security.core.CertificateValidationLoginModule
- com.sybase.security.ldap.LDAPLoginModule
- com.sybase.security.os.NTProxyLoginModule
- com.sybase.security.sap.SAPSSOTokenLoginModule
- com.sybase.security.core.CertificateAuthenticationLoginModule
- com.sybase.security.core.PreConfiguredUserLoginModule

- com.sybase.security.http.HttpAuthenticationLoginModule

com.sybase.security.core.NoSecLoginModule

The com.sybase.security.core.NoSecLoginModule package includes the following configurable properties:

Table 16. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 17. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 18. identity

Datatype	String
Default	nosec_identity
Required?	No
Requires server restart?	No
Read-only?	No

Table 19. implementationClass

Datatype	String
Default	com.sybase.security.businessobjects.NoSecLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 20. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 21. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 22. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 23. useFirstPass

Datatype	boolean
Default	FALSE

Required?	No
Requires server restart?	No
Read-only?	No

Table 24. useUsernameAsIdentity

Datatype	boolean
Default	TRUE
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.core.CertificateValidationLoginModule

The com.sybase.security.core.CertificateValidationLoginModule package contains the following configurable properties:

Table 25. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 26. implementationClass

Datatype	String
Default	com.sybase.security.core.CertificateValidationLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 27. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 28. validatedCertificateIsIdentity

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 29. enableRevocationChecking

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 30. trustedCertStore

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 31. trustedCertStorePassword

Datatype	String
Required?	No
Requires server restart?	No

Read-only?	No

Table 32. trustedCertStoreProvider

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 33. trustedCertStoreType

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 34. validateCertPath

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.ldap.LDAPLoginModule

The com.sybase.security.ldap.LDAPLoginModule package contains the following configurable properties:

Table 35. AuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 36. AuthenticationMethod

Datatype	String

Default	simple
Required?	No
Requires server restart?	No
Read-only?	No

Table 37. AuthenticationScope

Datatype	String (enumerated)
Alllowable values	• onelevel • subtree
Default	onelevel
Required?	No
Requires server restart?	No
Read-only?	No

Table 38. AuthenticationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 39. BindDN

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 40. BindPassword

Datatype	String
Required?	No
Requires server restart?	No

Read-only?	No

Table 41. CertificateAuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 42. DefaultSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 43. DigestMD5AuthenticationFormat

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 44. InitialContextFactory

Datatype	String
Default	com.sun.jndi.ldap.LdapCtxFactory
Required?	No
Requires server restart?	No
Read-only?	No

Table 45. ProviderURL

Datatype	String
Default	ldap://localhost:389
Required?	Yes

Requires server restart?	No
Read-only?	No

Table 46. Referral

Datatype	String (enumerated)
Allowable values	• ignore • follow • throw
Default	ignore
Required?	No
Requires server restart?	No
Read-only?	No

Table 47. RoleFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 48. RoleMemberAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 49. RoleNameAttributes

Datatype	String
Default	cn
Required?	No
Requires server restart?	No
Read-only?	No

Table 50. RoleScope

Datatype	String (enumerated)
Allowable values	• onelevel • subtree
Required?	No
Requires server restart?	No
Read-only?	No

Table 51. RoleSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 52. SecurityProtocol

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 53. SelfRegistrationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 54. SerializationKey

Datatype	String
Required?	No
Requires server restart?	No

Read-only?	No

Table 55. ServerType

Datatype	String (enumerated)
Allowable values	• sunone5 • msad2k • nsds4 • openldap
Required?	No
Requires server restart?	No
Read-only?	No

Table 56. UnmappedAttributePrefix

Datatype	String
Default	LDAP
Required?	No
Requires server restart?	No
Read-only?	No

Table 57. UseUserAccountControlAttribute

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 58. UserFreeformRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 59. UserRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 60. certificateAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 61. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 62. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 63. enableCertificateAuthentication

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 64. implementationClass

Datatype	String
Default	com.sybase.security.ldap.LDAPLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 65. ldapAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 66. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 67. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No

Read-only?	No

Table 68. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 69. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.os.NTProxyLoginModule

The com.sybase.security.os.NTProxyLoginModule package contains the following configurable properties:

Table 70. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 71. controlFlag

Datatype	String (enumerated)

Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 72. defaultAuthenticationServer

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 73. defaultDomain

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 74. extractDomainFromUsername

Datatype	boolean
Default	TRUE
Required?	No
Requires server restart?	No
Read-only?	No

Table 75. implementationClass

Datatype	String
Default	com.sybase.security.os.NTProxyLoginModule

Required?	Yes
Requires server restart?	No
Read-only?	No

Table 76. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 77. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 78. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 79. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No

Read-only?	No

com.sybase.security.sap.SAPSSOTokenLoginModule

The SAPSSOTokenLoginModule has been deprecated, Use the HttpAuthenticationLoginModule when SAP SSO2 token authentication is required. This authentication module will be removed in a future release.

The com.sybase.security.sap.SAPSSOTokenLoginModule package contains the following configurable properties:

Table 80. DisableServerCertificateValidation

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 81. SapSSOTokenPersistenceDataStore

Datatype	String
Default	jdbc/default
Required?	No
Requires server restart?	No
Read-only?	Yes

Table 82. SapServerCertificate

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 83. SapServerCertificatePassword

Datatype	String
Required?	No
Requires server restart?	No

Read-only?	No

Table 84. SapServerURL

Datatype	String
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 85. TokenExpirationInterval

Datatype	long
Default	120
Required?	No
Requires server restart?	No
Read-only?	No

Table 86. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 87. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No

Read-only?	No

Table 88. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.sap.SAPSSOTokenLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 89. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 90. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 91. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 92. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.core.CertificateAuthenticationLoginModule

The com.sybase.security.core.CertificateAuthenticationLoginModule package contains the following configurable properties:

Table 93. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 94. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 95. enableRevocationChecking

Datatype	boolean
Default	FALSE

Required?	No
Requires server restart?	No
Read-only?	No

Table 96. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.core.CertificateAuthenticationLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 97. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 98. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 99. trustedCertStore

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 100. trustedCertStorePassword

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 101. trustedCertStoreProvider

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 102. trustedCertStoreType

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 103. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 104. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 105. validateCertPath

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.core.PreConfiguredUserLoginModule

The com.sybase.security.core.PreConfiguredUserLoginModule package contains the following configurable properties:

Table 106. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 107. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 108. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.core.PreConfiguredUserLoginModule

Required?	Yes
Requires server restart?	No
Read-only?	No

Table 109. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 110. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 111. trustedCertStore

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 112. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 113. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 114. username

Datatype	String
Default	supAdmin
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 115. Password

Datatype	String
Default	“”
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 116. roles

Datatype	String
Default	SUP Administrator
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.http.HttpAuthenticationLoginModule

The com.sybase.security.core.PreConfiguredUserLoginModule package contains the following configurable properties:

Table 117. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.core.PreConfiguredUserLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 118. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 119. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 120. useFirstPass

Datatype	boolean

Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 121. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 122. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 123. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 124. URL

Datatype	String
Default	None
Required?	Yes

Requires server restart?	No
Read-only?	No

Table 125. DisableServerCertificateValidation

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 126. RolesHTTPHeader

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 127. SSOCookieNames

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Table 128. SuccessfulConnectionStatusCode

Datatype	int
Default	200
Required?	No
Requires server restart?	No
Read-only?	No

Authorization Provider

Supported authorizers.

- com.sybase.security.core.NoSecAuthorizer
- com.sybase.security.ldap.LDAPAuthorizer

com.sybase.security.core.NoSecAuthorizer

The com.sybase.security.core.NoSecAuthorizer package contains the following configurable properties:

Table 129. implementationClass

Datatype	String
Default	com.sybase.security.core.NoSecAuthorizer
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 130. providerType

Datatype	String
Default	Authorizer
Required?	Yes
Requires server restart?	No
Read-only?	Yes

com.sybase.security.ldap.LDAPAuthorizer

The com.sybase.security.ldap.LDAPAuthorizer package contains the following configurable properties:

Table 131. AuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 132. AuthenticationMethod

Datatype	String
Default	simple
Required?	No
Requires server restart?	No
Read-only?	No

Table 133. AuthenticationScope

Datatype	String (enumerated)
Alllowable values	• onelevel • subtree
Default	onelevel
Required?	No
Requires server restart?	No
Read-only?	No

Table 134. AuthenticationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 135. BindDN

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 136. BindPassword

Datatype	String

Required?	No
Requires server restart?	No
Read-only?	No

Table 137. CertificateAuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 138. DefaultSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 139. DigestMD5AuthenticationFormat

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 140. InitialContextFactory

Datatype	String
Default	com.sun.jndi.ldap.LdapCtxFactory
Required?	No
Requires server restart?	No
Read-only?	No

Table 141. ProviderURL

Datatype	String

Default	ldap://localhost:389
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 142. Referral

Datatype	String (enumerated)
Allowable values	• ignore • follow • throw
Default	ignore
Required?	No
Requires server restart?	No
Read-only?	No

Table 143. RoleFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 144. RoleMemberAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 145. RoleNameAttributes

Datatype	String
Default	cn
Required?	No

Requires server restart?	No
Read-only?	No

Table 146. RoleScope

Datatype	String (enumerated)
Allowable values	• onelevel • subtree
Required?	No
Requires server restart?	No
Read-only?	No

Table 147. RoleSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 148. SecurityProtocol

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 149. SelfRegistrationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 150. ServerType

Datatype	String (enumerated)

Allowable values	• sunone5 • msad2k • nsds4 • openldap
Required?	No
Requires server restart?	No
Read-only?	No

Table 151. UnmappedAttributePrefix

Datatype	String
Default	LDAP
Required?	No
Requires server restart?	No
Read-only?	No

Table 152. UseUserAccountControlAttribute

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 153. UserFreeformRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 154. UserRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No

Read-only?	No

Table 155. certificateAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 156. enableCertificateAuthentication

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 157. implementationClass

Datatype	String
Default	com.sybase.security.ldap.LDAPAuthorizer
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 158. ldapAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 159. providerType

Datatype	String
Default	Authorizer

Required?	Yes
Requires server restart?	No
Read-only?	Yes

Server Configuration

You can configure the following components through metadata:

- ReplicationSyncServer
- ReplicationNotifier_Push
- ReplicationNotifier_Pull
- MessagingSyncServer
- ConsolidatedDB
- AdministrationListener
- SecureAdministrationListener
- HTTPListener
- SecureHTTPListener
- SSLSecurityProfile
- KeyStore
- TrustStore
- JVM
- OCSP

Note: Properties you configure for an Unwired Server are cluster-affecting. Therefore, to make sure they are propagated correctly, Sybase recommends that you set them only on a primary cluster server.

ReplicationSyncServer

The `ReplicationSyncServer` component contains the following configurable properties:

Table 160. ml.cachesize

Datatype	String
Default	50M
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 161. ml.threadcount

Datatype	int
Default	5
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 162. sup.sync.certificate

Datatype	String
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 163. sup.sync.certificate_password

Datatype	String
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 164. sup.sync.httpsport

Datatype	int
Default	2481
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 165. sup.sync.port

Datatype	int
Default	2480
Required?	Yes
Requires server restart?	Yes

Read-only?	No

Table 166. sup.sync.protocol

Datatype	String
Allowable values	• http • https
Default	http
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 167. sup.user.options

Datatype	String
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 168. sup.sync.e2ee_type

Datatype	String
Allowable values	• rsa • ecc
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 169. sup.sync.e2ee_private_key

Datatype	String
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 170. sup.sync.e2ee_private_key_password

Datatype	String
Required?	No
Requires server restart?	Yes
Read-only?	No

ReplicationNotifier_Push

The `ReplicationNotifier_Push` component contains the following configurable properties:

Table 171. enable

Datatype	boolean
Default	TRUE
Required?	No
Requires server restart?	No
Read-only?	No

Table 172. poll_every

Datatype	String
Default	10s
Required?	Yes
Requires server restart?	No
Read-only?	No

ReplicationNotifier_Pull

The `ReplicationNotifier_Pull` component contains the following configurable properties:

Table 173. enable

Datatype	boolean
Default	TRUE
Required?	No

Requires server restart?	No
Read-only?	No

Table 174. poll_every

Datatype	String
Default	10s
Required?	Yes
Requires server restart?	No
Read-only?	No

MessagingSyncServer

Table 175. msg.admin.webservices.port

Datatype	int
Default	5100
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 176. msg.http.server.ports

Datatype	String
Default	5001,80
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 177. sup.msg.inbound_count

Datatype	int
Default	50
Required?	No
Requires server restart?	Yes

Read-only?	No

Table 178. sup.msg.inbound_queue_prefix

Datatype	String
Default	sup.mbs.
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 179. sup.msg.outbound_count

Datatype	int
Default	5
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 180. sup.msg.outbound_queue_prefix

Datatype	String
Default	sup.mbs.moca.
Required?	No
Requires server restart?	Yes
Read-only?	No

ConsolidatedDB

The `ConsolidatedDB` component contains the following configurable properties:

Table 181. cdb.asa.mode

Datatype	String
Default	primary
Required?	No
Requires server restart?	Yes

Read-only?	Yes

Table 182. cdb.databasename

Datatype	String
Default	default
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 183. cdb.dnsname

Datatype	String
Default	default-cdb
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 184. cdb.install_type

Datatype	String
Default	default
Required?	No
Requires server restart?	Yes
Read-only?	Yes

Table 185. cdb.password

Datatype	String
Default	sql
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 186. cdb.serverhost

Datatype	String
Default	gma
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 187. cdb.servername

Datatype	String
Default	gma_primary
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 188. cdb.serverport

Datatype	int
Default	5200
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 189. cdb.threadcount

Datatype	int
Default	20
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 190. cdb.type

Datatype	String

Allowable values	• Sybase_ASA • Sybase_ASE
Default	Sybase_ASA
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 191. cdb.user.options

Datatype	String
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 192. cdb.username

Datatype	String
Default	dba
Required?	Yes
Requires server restart?	Yes
Read-only?	No

AdministrationListener

The `AdministrationListener` component contains the following configurable properties:

Table 193. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 194. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 195. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 196. sup.socket.listener.maxthreads

Datatype	int
Default	100
Required?	Yes
Requires server restart?	Yes
Read-only?	No

SecureAdministrationListener

The `SecureAdministrationListener` component contains the following configurable properties:

Table 197. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 198. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 199. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 200. sup.socket.listener.security.profile

Datatype	String
Default	default
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 201. sup.socket.listener.maxthreads

Datatype	int
Default	100
Required?	Yes
Requires server restart?	Yes
Read-only?	No

HTTPListener

The `HTTPListener` component contains the following configurable properties:

Table 202. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 203. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 204. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 205. sup.socket.listener.maxthreads

Datatype	int
Default	100
Required?	Yes
Requires server restart?	Yes
Read-only?	No

SecureHTTPListener

The `SecureHTTPListener` component contains the following configurable properties:

Table 206. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 207. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 208. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 209. sup.socket.listener.security.profile

Datatype	String
Default	default
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 210. sup.socket.listener.maxthreads

Datatype	int
Default	100
Required?	Yes
Requires server restart?	Yes
Read-only?	No

SSLSecurityProfile

The `SSLSecurityProfile` component contains the following configurable properties:

Table 211. sup.security.profile.auth

Datatype	String
Allowable values	• intl • intl_mutual • strong • strong_mutual • domestic • domestic_mutual
Default	intl
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 212. sup.security.profile.key.alias

Datatype	String
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 213. sup.security.profile.name

Datatype	String

Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

KeyStore

The `KeyStore` component contains the following configurable properties:

Table 214. sup.sync.sslkeystore

Datatype	String
Default	Repository/Security/keystore.jks
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 215. sup.sync.sslkeystore_password

Datatype	String
Default	changeit
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

TrustStore

The `TrustStore` component contains the following configurable properties:

Table 216. sup.sync.ssltruststore

Datatype	String
Default	Repository/Security/truststore.jks
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 217. sup.sync.ssltruststore_password

Datatype	String
Default	changeit
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

JVM

The JVM component contains the following configurable properties:

Table 218. DJC_JVM_MINHEAP

Datatype	String
Default	64M
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 219. DJC_JVM_MAXHEAP

Datatype	String
Default	256M
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 220. DJC_JVM_STACKSIZE

Datatype	String
Default	400K
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 221. DJC_JVM_USEROPTIONS

Datatype	String
Default	""
Required?	No
Requires server restart?	Yes
Read-only?	No

OCSP

The `OCSP (Online Certificate Status Protocol)` component contains the following configurable properties:

Table 222. ocsp.enable

Datatype	Boolean
Default	False
Required?	No
Requires server restart?	No
Read-only?	No

Table 223. ocsp.responderURL

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Table 224. ocsp.responderCertIssuerName

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Table 225. ocsp.responderCertSerialNumber

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Table 226. ocsp.responderCertSubjectName

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Server Log Configuration

You can perform log configuration through the `LocalFileAppender` log appenders. The log appender can contain one or more of the following log buckets:

- MSG
- Trace
- MMS
- Security
- Mobilink
- DataServices
- Proxy
- Other

LocalFileAppender

The `LocalFileAppender` log appender contains the following configurable properties:

Table 227. LogLevel

Datatype	String

Allowable values	• TRACE • DEBUG • INFO • WARN • ERROR • OFF
Default	WARN
Required?	No
Requires server restart?	No
Read-only?	No

Table 228. async

Datatype	boolean
Default	FALSE
Required?	No
Requires Server Re-start?	No
Read Only?	No

Table 229. dateRollover

Datatype	String
Allowable Values	• NONE • HOURLY • DAILY • WEEKLY • MONTHLY • YEARLY
Default	NONE
Required?	No
Requires Server Re-start?	No
Read Only?	No

Table 230. filename

Datatype	String
Default	null
Required?	Yes
Requires Server Restart?	No
Read Only?	No

Table 231. maximumRolloverFiles

Datatype	int
Default	1
Required?	No
Requires Server Restart?	No
Read Only?	No

Table 232. sizeRollover

Datatype	String
Default	10mb
Required?	No
Requires Server Restart?	No
Read Only?	No

Property Reference

Review properties of the administration client API.

Application Connection Properties

Application Connection properties fall into various categories.

- Apple Push Notifications
- Application Settings
- BlackBerry Push Notifications
- Connection
- Custom Settings
- Device Advanced
- Device Info
- Proxy
- Security Settings
- User registration

Apple Push Notification Properties

Apple Push Notification properties allow iPhone users to install messaging client software on their devices. This requires you to create different e-mail activation messages using the appropriate push notification properties.

ID: property name (type)	Description	Default
2600: Enable (boolean)	Enables if push notification using APNs is enabled or not.	True
2601: Alert (boolean)	Use the iOS standard alert.	True
2602: Badges (boolean)	Use the badge of the application icon.	True
2603: Sounds (boolean)	Use a if a sound is a made when a notification is received. The sound files must reside in the main bundle of the client application. Because custom alert sounds are played by the iOS system-sound facility, they must be in one of the supported audio data formats. See the iOS developer documentation.	True

ID: property name (type)	Description	Default
2604: Delivery Threshold (integer)	The frequency, in minutes, with which groupware notifications are sent to the device. Valid values: 0 – 65535.	1
2605: Alert Message (string)	The message that appears on the client device when alerts are enabled.	New items available
2606: APNS Device Token (string)	The Apple push notification service token. An application must register with Apple push notification service for the iOS to receive remote notifications sent by the application's provider. After the device is registered for push properly, this should contain a valid device token. See the iOS developer documentation.	n/a

Application Settings Properties

Application settings display details that identify the Application Identifier, Domain, Security Configuration of an application connection template.

ID: property name (type)	Description	Default
Domain	The domain selected for the connection template.	
Security Configuration	The security configuration defined for the connection template.	
Automatic Registration Enabled	The value is set to **True** when the application connection registration is carried out automatically.	
Application Identifier	The application identifier registered on SCC.	

BlackBerry Push Notification Properties

BlackBerry push notification properties allow BlackBerry users to install messaging client software on their devices.

Property	Description
Enabled	Enables notifications to the device if the device is offline. This feature sends a push notification over an IP connection only long enough to complete the Send/Receive data exchange. BlackBerry Push notifications overcome issues with always-on connectivity and battery life consumption over wireless networks. Acceptable values: true (enabled) and false (disabled). If this setting is false, all other related settings are ignored. Default: true

Property	Description
Delivery threshold	The minimum amount of time the server waits to perform a push notification to the device since the previous push notification (in minutes). This controls the maximum number of push notifications sent in a given time period. For example, if three push notifications arrive 10 seconds apart, the server does not send three different push notifications to the device. Instead they are sent as a batch with no more than one push notification per X minutes (where X is the delivery threshold). Acceptable values: 0 – 65535. Default: 1
Push listener port	The push listener port reported by the device on which it listens for notifications. This port is automatically assigned by the client. For example, if there is another application already listening on this port, a free port is searched for. Default: 5011
Device PIN	Every Blackberry device has a unique permanent PIN. During initial connection and settings exchange, the device sends this information to the server. Unwired Server uses this PIN to address the device when sending notifications, by sending messages through the BES/MDS using an address such as: Device="Device PIN" + Port="Push Listener port". Default: 0
Name	The BES server to which this device's notifications are sent. In cases where there are multiple BES servers in an organization, define all BES servers.

Connection Properties

Connection properties define the connection information used by Unwired Server to relate a user to a device.

ID: property name (type)	Description	Default
1: Server Name (string)	The DNS name or IP address of the Unwired Server, such as "myserver.mycompany.com". If using Relay Server, the server name is the IP address or fully qualified name of the Relay Server host.	n/a
2: Server Port(integer)	The port used for messaging connections between the device and Unwired Server. If using Relay Server, this is the Relay Server port.	5001
3: Farm ID (string)	The string associated with the Relay Server farm ID. Can contain only letters A – Z (uppercase or lowercase), numbers 0 – 9, or a combination of both.	0

ID: property name (type)	Description	Default
6: Activation Code (string)	The original code sent to the user in the activation e-mail. Can contain only letters A – Z (uppercase or lowercase), numbers 0 – 9, or a combination of both. Acceptable range: 1 to 10 characters.	n/a

Custom Settings Properties

Define one of four available custom strings that are retained during reregistration and cloning.

Change the property name and value according to the custom setting you require. The custom settings can be of variable length, with no practical limit imposed on the values. You can use these properties to either manually control or automate how workflow-related messages are processed:

- Manual control – an administrator can store an employee title in one of the custom fields. This allows employees of a specific title to respond to a particular message.
- Automated – a developer stores the primary key of a back-end database using a custom setting. This key allows the database to process messages based on messaging device ID.

ID: property name (type)	Description	Default
2300: Custom 1(string)	A custom string which is retained during reregistration and cloning.	n/a
2301: Custom 2(string)	A custom string which is retained during reregistration and cloning.	n/a
2302: Custom 3(string)	A custom string which is retained during reregistration and cloning.	n/a
2303: Custom 4(string)	A custom string which is retained during reregistration and cloning.	n/a

Device Information Properties

Information properties display details that identify the mobile device, including International Mobile Subscriber identity (IMSI), phone number, device subtype, and device model.

ID: property name (type)	Description	Default
1200: Model (string)	The manufacturer of the registered mobile device.	n/a

ID: property name (type)	Description	Default
1201: Device Subtype (string)	The device subtype of the messaging device. For example, if the device model is a BlackBerry, the subtype is the form factor (for example, BlackBerry Bold).	n/a
1202: Phone Number (string)	The phone number associated with the registered mobile device.	n/a
1203: IMSI (string)	The International Mobile Subscriber identity, which is a unique number associated with all Global System for Mobile communication (GSM) and Universal Mobile Telecommunications System (UMTS) network mobile phone users. To locate the IMSI, check the value on the SIM inside the phone.	n/a

Advanced Device Properties

Advanced properties set specific behavior for messaging devices.

ID: property name (type)	Description	Default
1300: Keep Alive (sec) (integer)	The Keep Alive frequency used to maintain the wireless connection, in seconds. Acceptable values: 30 to 1800.	240
1301: Device Log Items(integer)	The number of items persisted in the device status log. Acceptable values: 5 to 100.	50
1302: Debug Trace Level (integer)	The amount of detail to record to the device log. Acceptable values: 1 to 5, where 5 has the most level of detail and 1 the least.	1
1303: Debug Trace Size (KB) (integer)	The size of the trace log on the device (in KB). Acceptable values: 50 to 10,000.	50
1304: Allow Roaming (boolean)	Use ifdevice is allowed to connect to server while roaming. Acceptable values: true and false.	True
1305: Relay Server URL Prefix (string)	The URL prefix to be used when the device client is connecting through Relay Server. The prefix you set depends on whether Relay Server is installed on IIS or Apache. Acceptable values: • For IIS – use `/ias_relay_server/client/rs_client.dll`. • For Apache – use `/cli/iasrelayserver`.	n/a

Proxy Properties

Proxy properties define parameters to connect Relay Server Outbound Enabler to a Relay Server through a proxy server.

ID: property name (type)	Description	Default
Application Endpoint	The application endpoint.	n/a
Push Endpoint	The URL for the push endpoint.	http://<server_hostname>:8000/GWC/SUPNotification

Security Settings Properties

Security settings display the device security configuration.

ID: property name (type)	Description	Default
E2E Encryption Enabled	Indicates whether end-to-end encryption is enabled.	
E2E Encryption Type	Indicates the asymmetric cipher used for key exchange for end-to-end encryption, either RSA or ECC.	
TLS Type	Indicates the TLS type for device to Unwired Server communication, either RSA or ECC.	

User Registration Properties

Device user registration properties allow you to customize the registration request that is delivered to the device.

ID: property name (type)	Description	Default
900: Activation code length (integer)	The number of characters to be contained in the activation code. Acceptable values: 1 to 10.	3

ID: property name (type)	Description	Default
901: Activation code expiration (hours) (integer)	Defines how long a user has to activate their account, in hours, before the account activation period expires. Acceptable values: 1 to 10,000 hours.	72

EIS Data Source Connection Properties Reference

Name and configure connection properties when you create connection pools in Sybase Control Center to enterprise information systems (EIS) .

JDBC Properties

Configure Java Database Connectivity (JDBC) connection properties.

This list of properties can be used by all datasource types. Sybase does not document native properties used only by a single driver. However, you can also use native driver properties, naming them using this syntax:

```
<driver_type>:<NativeConnPropName>=<SupportedValue>
```

Note: If Unwired Server is connecting to a database with a JDBC driver, ensure you have copied required JAR files to correct locations. See the *Installation for Runtime* guide.

Name	Description	Supported values
afterInsert	Changes the value to `into` if a database requires `insert into` rather than the abbreviated `into`.	`into`
batchDelimiter	Sets a delimiter, for example, a semicolon, that can be used to separate multiple SQL statements within a statement batch.	`<delimiter>`
blobUpdater	Specifies the name of a class that can be used to update database BLOB (long binary) objects when the BLOB size is greater than psMaximumBlobLength.	`<class name>` The class must implement the `com.sybase.djc.sql.BlobUpdater` interface.

Name	Description	Supported values
compactColumnAlias	An expression that uses the nested variables "${index}" and "${column}" for shortening column names in result sets. This can reduce the data transmitted between the database server and the application server.	An expression. For example: `_${index}=${column} ${column} AS _${index}`
clobUpdater	Specifies the name of a class that can be used to update database CLOB (long string) objects when the CLOB size is greater than psMaximumClobLength.	`<class name>` The class must implement the `com.sybase.djc.sql.ClobUpdater` interface.
codeSet	Specifies how to represent a repertoire of characters by setting the value of CS_SYB_CHARSET for this datasource. Used when the data in the datasource is localized. If you do not specify the correct code set, characters may be rendered incorrectly.	`[server]` If the value is server, the value of the current application server's defaultCodeSet property is used.

Name	Description	Supported values
commitProtocol	Specifies how Unwired Server handles connections for a datasource at commit time, specifically when a single transaction requires data from multiple endpoints. If you use XA, the recovery log is stored in the tx_manager datasource, and its commit protocol must be optimistic. If tx_manager is aliased to another datasource (that is, one that is defined with the alias-For property), the commit protocol for that datasource must be optimistic. A last-resource optimization ensures full conformance with the XA specification. The commit protocol for all other datasources should be XA_2PC. Alternately, a transaction that accesses multiple datasources for which the commit protocols are optimistic is permitted.	`[optimistic| pessimistic | XA_2PC]` Choose only one of these protocols: • Optimistic – enables connections to be committed without regard for other connections enlisted in the transaction, assuming that the transaction is not marked for rollback and will successfully commit on all resources. Note: if a transaction accesses multiple data sources with commit protocol of "optimistic", atomicity is not guaranteed. • Pessimistic – specifies that you do not expect any multi-resource transactions. An exception will be thrown (and transaction rolled back) if any attempt is made to use more than one "pessimistic" data source in the same transaction. • XA_2PC – specifies use of the XA two phase commit protocol. If you are using two phase commit, then the recovery log is stored in the "tx_manager" data source, and that data source (or the one it is aliased to) must have the commit protocol of "optimistic" or "pessimistic". All other data sources for which atomicity must be ensured should have the "XA_2PC" commit protocol.

Name	Description	Supported values
dataSourceClass	Sets the class that implements the JDBC datasource. Use this property (along with the driverClass property) only if you do not have a predefined database-type entry in Unwired Server for the kind of SQL database you are connecting to. For example, you must use this property for MySQL database connections. You can implement a datasource class to work with a distributed transaction environment. Because Unwired Server supports distributed transactions, some datasources may require that a datasource class be implemented for Unwired Server to interact with it. For two-phase transactions, use the xaDataSourceClass connection property instead.	`<com.`*`mydata-source`*`.jdbc.Driver>`
databaseCommandEcho	Echoes a database command to both the console window and the server log file. Use this property to immediately see and record the status or outcome of database commands. When you enable this property, Unwired Server echoes every SQL query to `ml.log`, which may help you debug your application.	`[true|false]` Set a value of 1 to echo the database commands like `databaseStartCommand`, and `databaseStopCommand`. Otherwise, do not set this property, or use a value of 0 to disable the echo.

Name	Description	Supported values
databaseCreateCommand	Specifies the operating system command used to create the database for this datasource. If this command is defined and the file referenced by ${databaseFile} does not exist, the command is run to create the database when an application component attempts to obtain the first connection from the connection pool for this datasource.	`<command>` Example: `<UnwiredPlatform_InstallDir>\Servers\SQLAnywhere11\BIN32\dbinit -q ${databaseFile}`
databaseFile	Indicates the database file to load when connecting to a datasource. Use this property when the path to the database file differs from the one normally used by the database server. If the database you want to connect to is already running, use the databaseName connection parameter.	`<string>` Supply a complete path and file name. The database file you specify must be on the same host as the server.

Name	Description	Supported values
databaseName	Identifies a loaded database with which to establish a connection, when connecting to a datasource. Set a database name, so you can refer to the database by name in other property definitions for a datasource. If the database to connect to is not already running, use the databaseFile connection parameter so the database can be started. **Note:** For Unwired Server, you typically do not need to use this property. Usually, when you start a database on a server, the database is assigned a name. The mechanism by which this occurs varies. An administrator can use the DBN option to set a unique name, or the server may use the base of the file name with the extension and path removed.	`[DBN|default]` If you set this property to default, the name is obtained from the DBN option set by the database administrator. If no value is used, the database name is inherited from the database type.
databaseStartCommand	Specifies the operating system command used to start the database for this datasource. If this command is defined and the database is not running, the command is run to start the database when the datasource is activated.	`<command>` Example: `<UnwiredPlatform_InstallDir>\Servers\SQLAnywhere11\BIN32\dbsrv11.exe`

databaseStopCommand	Specifies the operating system command used to stop the database for this datasource. If this property is defined and the database is running, this command executes during shutdown.	`<command>`
		
		For a Adaptive Server® Anywhere database, where the user name and password are the defaults (dba and sql), enter:
		
		`<UnwiredPlatform_InstallDir>\Servers\SQLAnywhere11\BIN32\dbsrv11.exe`

Name	Description	Supported values
databaseType	Specifies the database type.	`<database type>`
databaseURL	Sets the JDBC URL for connecting to the database if the datasource requires an Internet connection. Typically, the server attempts to construct the database URL from the various connection properties you specify (for example, portNumber, databaseName). However, because some drivers require a special or unique URL syntax, this property allows you to override the server defaults and instead provide explicit values for this URL.	`<JDBCurl>` The database URL is JDBC driver vendor-specific. For details, refer to the driver vendor's JDBC documentation.
disableAutoCommit	Enables or disables calling auto-commit mode. Auto-commit means that every update to the database is immediately made permanent.	`[true|false]` The default is false.
disablePrefetch	Enables or disables prefetch. Prefetch optimizes container-managed persistence by batching queries from a parent to its children (for example, from a customer to orders), to reduce the calls from the application server to the database.	`[true|false]` The default is true.
disableTriggers	Select to deactivate database triggers, on a per-connection basis, when the application server accesses the database. If selected, the database must support both the `set triggers on` and `set triggers off` commands.	`[true|false]` The default is false.

Name	Description	Supported values
driverClass	Sets the name of the class that implements the JDBC driver. Use this property (along with the dataSourceClass property) only if you do not have a predefined database-type entry in Unwired Server for the kind of SQL database you are connecting to. For example, MySQL database connections require you to use this connection property. To create a connection to a database system, you must use the compatible JDBC driver classes. Sybase does not provide these classes; you must obtain them from the database manufacturer.	*<Class.forName("foo.bar.Driver")>* Replace <Class.forName("foo.bar.Driver")> with the name of your driver.
driverDebug	Enables debugging for the driver.	`[true|false]` Set to true to enable debugging, or false to disable.
driverDebugSettings	Configures debug settings for the driver debugger.	`[default|<setting>]` The default is STATIC:ALL.
endpointName	The JDBC datasource name.	JDBC datasource name.
getDateAndTime	A SQL query to get the date and time.	A valid SQL query. The default is `select getdate()`.

Name	Description	Supported values
InitialPoolSize	Sets the initial number of connections in the pool for a datasource. In general, holding a connection causes a less dramatic performance impact than creating a new connection. Keep your pool size large enough for the number of concurrent requests you have; ideally, your connection pool size should ensure that you never run out of available connections. The initialPoolSize value is applied to the next time you start Unwired Server.	`<int>` Replace <int> with an integer to preallocate and open the specified number of connections at start-up. The default is 0. Sybase suggests that you start with 0, and create additional connections as necessary. The value you choose allows you to create additional connections before client synchronization requires the server to create them.
isDownloadZipped	Specifies whether the driver file downloaded from jdbcDriverDownloadURL is in `.ZIP` format. This property is ignored if the value of jdbcDriverDownloadURL connection is an empty string.	`[True|False]` The default is false. The file is copied, but not zipped to `<UnwiredPlatform-install>\lib\jdbc`. Set isDownloadZipped to true to save the file to `<UnwiredPlatform-install>\lib\jdbc` and unzip the archived copy.
jdbc:DISABLE_UNPROCESSED_PARAM_WARNINGS	All properties starting with “jdbc:” are used to pass the suffix (such as DISABLE_UNPROCESSED_PARAM_WARNINGS) to the JDBC driver while getting a connection. This property is used for the jConnect driver. Set this property to true can disable the warning of “An output parameter was received and ignored”.	`[True|False]` The default is false. This property is for Sybase ASA or Sybase ASE databases only.

Name	Description	Supported values
jdbc:IS_CLOSED_TEST	As above, this property is used for the jConnect driver. You can force jConnect to follow the standard JDBC behavior for `isClosed()` by setting the IS_CLOSED_TEST connection property to the special value 'INTERNAL'. The INTERNAL setting means that jConnect returns true for `isClosed()` only when `Connection.close()` has been called, or when jConnect has detected an IOException that has disabled the Connection. You can specify a query other than `sp_mda` to use when `isClosed()` is called. For example, if you want jConnect to try *select 1* when `isClosed()` is called, you can set the IS_CLOSED_TEST connection property to *select 1*.	The default is INTERNAL.
jdbc:DriverType	The driverType property to be passed to the JDBC driver class. For example, for Oracle, you can set this property to “thin”.	The driverType property. For an Oracle database type, use "thin".
jdbcDriverDownloadURL	Specifies the URL from which you can download a database driver. Use this property with isDownloadZipped to put the driver in an archive file before the download starts.	`<URL>` Replace <URL> with the URL from which the driver can be downloaded.
jit:imageParameterType	Defines the SQL type of the image parameter. All properties that start with “jit:” are used for the Sybase JIT DataSource only.	A varbinary (16384) value. For example, varbinary(255).
jit:textParameterType	Defines the SQL type of the text parameter. Used for the Sybase JIT DataSource only.	A varchar (16384) value.

Name	Description	Supported values
jit:unitextParameterType	Defines the SQL type of the unicode text parameter. Used for the Sybase JIT DataSource only.	A univarchar (16384) value.
language	For those interfaces that support localization, this property specifies the language to use when connecting to your target database. When you specify a value for this property, Unwired Server: • Allocates a CS_LOCALE structure for this connection • Sets the CS_SYB_LANG value to the language you specify • Sets the Microsoft SQL Server CS_LOC_PROP connection property with the new locale information Unwired Server can access Unicode data in an Adaptive Server® 12.5 or later, or in Unicode columns in Adaptive Server 12.5 or later. Unwired Server automatically converts between double-byte character set (DBCS) data and Unicode, provided that the Language and CodeSet parameters are set with DBCS values.	`<language>` Replace <language> with the language being used.
maxIdleTime	Specifies the number of seconds an idle connection remains in the pool before it is dropped.	`<int>` If the value is 0, idle connections remain in the pool until the server shuts down. The default is 60.

Name	Description	Supported values
maxPoolSize	Sets the maximum number of connections allocated to the pool for this datasource. Increase the maxPoolSize property value when you have a large user base. To determine whether a value is high enough, look for ResourceMonitorTimeoutException exceptions in `<hostname>-server.log`. Continue increasing the value, until this exception no longer occurs. To further reduce the likelihood of deadlocks, configure a higher value for maxWaitTime. To control the range of the pool size, use this property with minPoolSize.	`<int>` A value of 0 sets no limit to the maximum connection pool size.
maxWaitTime	Sets the maximum number of seconds to wait for a connection before the request is cancelled.	`<int>` The default is 60.
maxStatements	Specifies the maximum number of JDBC prepared statements that can be cached for each connection by the JDBC driver. The value of this property is specific to each JDBC driver.	`<int>` A value of 0 (default) sets no limit to the maximum statements.
minPoolSize	Sets the minimum number of connections allocated to the pool for this datasource.	`<int>` A value of 0 (default) sets no limit to the minimum connection pool size.

Name	Description	Supported values
networkProtocol	Sets the protocol used for network communication with the datasource. Use this property (along with the driverClass, and dataSourceClass properties) only if you do not have a predefined database-type entry in Unwired Server for the kind of SQL database you are connecting to. For example, you may be required to use this property for MySQL database connections.	The network protocol is JDBC driver vendor-specific. There are no predefined values. See the driver vendor's JDBC documentation.
ownerPrefix	The owner prefix for stored procedures and table names in this datasource. A prefix is used by the EJB persistence manager and JIT driver wrappers to qualify database identifiers for stored procedures and tables.	An owner prefix.
password	Specifies the password for connecting to the database.	`[default|<password>]`
pingAndSetSessionAuth	Runs the ping and session-authorization commands in a single command batch; may improve performance. You can only enable the Ping and Set Session Auth property if you have enabled the Set Session Auth property so database work runs under the effective user ID of the client.	`[True|False]` Set to true to enable, or false to disable.
pingConnections	Pings connections before attempting to reuse them from the connection pool.	`[True|False]` Set to true to enable ping connections, or false to disable.
pingSQL	Specify the SQL statement to use when testing the database connection with ping.	`[default|<statement>]` Replace <statement> with the SQL statement identifier. The default is "select 1".

Name	Description	Supported values
portNumber	Sets the server port number where the database server listens for connection requests.	`[default|<port>]` Replace <port> with the TCP/IP port number to use (that is, 1 – 65535). If you set the value as default, the default protocol of the datasource is used.
psMaximumBlobLength	Indicates the maximum number of bytes allowed when updating a BLOB datatype using PreparedStatement.setBytes.	`[default|<int>]` Replace <int> with the number of bytes allowed during an update. The default is 16384.
psMaximumClobLength	Indicates the maximum number of characters allowed when updating a CLOB datatype using PreparedStatement.setString.	`[default|<int>]` Replace <int> with the number of bytes allowed during an update. The default is 16384.
roleName	Sets the database role that the user must have to log in to the database.	`[default|<name>]` If you set this value to default, the default database role name of the datasource is used.
selectWithSharedLock	A template SQL statement for selecting rows and acquiring a shared lock. If your database server does not support shared locks, specify a template for acquiring exclusive locks.	A template SQL statement. For example, for a Sybase ASA database type: `${selectList}${intoClause}${fromClause} holdlock${whereClause}`

selectWithUpdateLock	A template SQL statement for selecting rows and acquiring an exclusive lock. The configuration property name is `selectWithUpdateLock`. If your database server does not support exclusive locks, specify a template for acquiring shared locks.	A template SQL statement.
		For example, for a Sybase ASA database type:
		`update ${mainTable} set ${touchColumn} = 1 -${touchColumn}${fromClause}${whereClause};;`
		`${selectList}${intoClause}${fromClause}${whereClause}`

Name	Description	Supported values
serializableSelect	A template SQL statement for selecting rows and acquiring a lock that ensures strict serializability, in terms of equivalence with serial schedules.	A template SQL statement. For example, for a Sybase database type: `${selectList}${intoClause}${fromClause} holdlock${whereClause}`
serverName	Defines the host where the database server is running.	`<name>` Replace <name> with an appropriate name for the server.
serviceName	Defines the service name for the datasource. For SQL Anywhere servers, use this property to specify the database you are attaching to.	`<name>` Replace <name> with an appropriate name for the service.
setSessionAuth	Establishes an effective database identity that matches the current mobile application user. If you use this property, you must also use setSessionAuthSystemID to set the session ID. Alternately you can pingAndSetSessionAuth if you are using this property with pingConnection. The pingAndSetSessionAuth property runs the ping and session-authorization commands in a single command batch, which may improve performance.	`[true|false]` Choose a value of 1 to use an ANSI SQL set session authorization command at the start of each database transaction. Set to 0 to use session-based authorizations.
setSessionAuthSystemID	If Set Session Authorization is enabled, specifies the database identity to use when the application server accesses the database from a transaction that runs with "system" identity.	`<database identity>` Replace <database identity> with the database identifier.

Name	Description	Supported values
startWait	Sets the wait time (in seconds) before a connection problem is reported. If the start command completes successfully within this time period, no exceptions are reported in the server log. startWait time is used only with the databaseStartCommand property.	`<int>` Replace <int> with the number of seconds Unwired Server waits before reporting an error.
truncateNanos	Sets a divisor/multiplier that is used to round the nanoseconds value in a java.sql.Timestamp to a granularity that the DBMS supports.	`[default|<int>]` The default is 10 000 000.
useQuotedIdentifiers	Specifies whether or not SQL identifiers are quoted.	`[True|False]` Set to true to enable use of quoted identifiers, or false to disable.
useTransactionalPing	Enables or disables the attempt to ping a connection from within a new transaction.	`[True|False]` The default is true.
user/User	Identifies the user who is connecting to the database.	`[default|<user name>]` Replace <user name> with the database user name. For DB2 and SQL Server databases, this property is `user`. For Informix, Oracle, and SQL Anywhere databases, this property is `User`.
xaDataSourceClass	Specifies the class name or library name used to support two-phase commit transactions, and the name of the XA resource library.	`<class name>` Replace <class name> with the class or library name. • SQL Anywhere database: `com.sybase.jdbc3.jdbc.SybXADataSource` • Oracle database: `oracle.jdbc.xa.client.OracleXADataSource`

SAP Java Connector Properties

Configure SAP Java Connector (JCo) connection properties.

For a comprehensive list of SAP JCo properties you can use to create an instance of a client connection to a remote SAP system, see *http://help.sap.com/javadocs/NW04/current/jc/com/sap/mw/jco/JCO.html#createClient(java.util.Properties)*.

Table 233. General connection parameters

Name	Description	Supported values
endpointName	Specifies the endpoint name.	Endpoint name
jco.client.alias_user	Specifies the alias user name.	Alias user name.
jco.client.client	Specifies the SAP client.	Three-digit client number; preserve leading zeros if they appear in the number
jco.client.user	Specifies the login user ID.	User name for logging in to the SAP system If using X.509 certificate authentication, remove the JCo properties `jco.client.passwd` and `jco.client.user` defined for the SAP connection profile in Sybase Control Center (SCC).
jco.client.passwd	Specifies the login password.	Password for logging in to the SAP system
jco.client.lang	Specifies a login language.	ISO two-character language code (for example, EN, DE, FR), or SAP-specific single-character language code. As a result, only the first two characters are ever used, even if a longer string is entered. The default is EN.
jco.client.sysnr	Indicates the SAP system number.	SAP system number
jco.client.ashost	Identifies the SAP application server.	Host name of a specific SAP application server
jco.client.mshost	Identifies the SAP message server.	Host name of the message server

Name	Description	Supported values
jco.client.gwhost	Identifies the SAP gateway host.	Host name of the SAP gateway Example: `GWHOST=hs0311`
jco.client.gwserv	Identifies the SAP gateway service.	Service name of the SAP gateway Example: `GWSERV=sapgw53`
jco.client.idle_timeout	Specifies the idle timeout, in seconds, for the connection after which it will be closed by R/3. Only positive values are allowed.	Idle timeout, in seconds.
jco.client.r3name	Specifies R/3 name.	Name of the SAP system
jco.client.group	Identifies the group of SAP application servers.	Group name of the application servers
jco.client.tpname	Identifies the program ID of the external server program.	Path and name of the external RFC server program, or program ID of a registered RFC server program Example: `TPNAME=/sap/srfcserv`
jco.client.tphost	Identifies the host of the external server program. This information determines whether the RFC client connects to an RFC server started by the SAP gateway or to an already registered RFC server. **Note:** If the gateway host and external server program host are different, make sure that the SAP gateway has access to start the server program through a remote shell.	Host name of the external RFC server program Example: `TPHOST=hs0311`
jco.client.type	Identifies the type of remote host.	2: R/2 3: R/3 E: external
jco.client.trace	Specifies whether or not to enable RFC trace.	0: disable 1: enable

Name	Description	Supported values
jco.client.codepage	Identifies the initial code page in SAP notation. A code page is used whenever character data is processed on the application server, appears on the front end, or is rendered by a printer.	Four-digit SAP code page number
jco.client.abap_debug	Enables or disables ABAP debugging. If enabled, the connection is opened in debug mode and the invoked function module can be stepped through in the debugger. For debugging, an SAP graphical user interface (SAPGUI) must be installed on the same machine the client program is running on. This can be either a normal Windows SAPGUI or a Java GUI on Linux/UNIX systems.	0: no debugging 1: attach a visible SAPGUI and break at the first ABAP statement of the invoked function module
jco.client.use_sapgui	Specifies whether a remote SAP graphical user interface (SAPGUI) should be attached to the connection. Some older BAPIs need an SAPGUI because they try to send screen output to the client while executing.	0: no SAPGUI 1: attach an "invisible" SAPGUI, which receives and ignores the screen output 2: attach a visible SAPGUI For values other than 0 a SAPGUI needs to be installed on the machine, where the client program is running. This can be either a Windows SAPGUI or a Java GUI on Linux/Unix systems.
jco.client.getsso2	Generates an SSO2 ticket for the user after login to allow single sign-on. If RfcOpenConnection() succeeds, you can retrieve the ticket with RfcGetPartnerSSOTicket() and use it for additional logins to systems supporting the same user base.	0: do not generate SSO2 ticket 1: generate SSO2 ticket

Name	Description	Supported values
jco.client.mysapsso2	Indicates whether or not to use the specified SAP Cookie Version 2 (SSO2) as the login ticket instead of user ID and password.	User: $MYSAPSSO2$ Password: Base64-encoded ticket Login with single sign-on is based on secure network connection (SNC) encryption and can only be used in combination with an SNC.
jco.client.x509cert	Indicates whether or not to use the specified X509 certificate as the login certificate instead of user ID and password.	User: $X509CERT$ Password: Base64-encoded ticket Login with X509 is based on secure network connection (SNC) encryption and can only be used in combination with an SNC.
jco.client.lcheck	Enables or disables login check at open time.	0: disable 1: enable If you set this to 0, RfcOpenConnection() opens a network connection, but does not perform the login procedure. Therefore, no user session is created inside the back-end system. This parameter is intended only for executing the function module RFC_PING.
jco.client.grt_data	Provides additional data for graphical user interface (GUI) to specify the SAProuter connection data for the SAPGUI when it is used with RFC.	/H/ *`router string`* : the entire router string for the SAPGUI /P/ *`password`* : specify this value if the password for the SAPGUI connection is not the same as the password for the RFC connection.
jco.client.use_guihost	Identifies which host to redirect the remote graphical user interface to.	Host name
jco.client.use_guiserv	Identifies which service to redirect the remote graphical user interface to.	Name of the service
jco.client.use_guiprogid	Indicates the program ID of the server that starts the remote graphical user interface.	Program ID of the server

Name	Description	Supported values
jco.client.snc_mode	Enables or disables secure network connection mode.	0: off 1: on
jco.client.snc_partner-name	Identifies the secure network connection partner.	Secure network connection name of the application server (for example, p:CN=R3, O=XYZ-INC, C=EN)
jco.client.snc_qop	Specifies the secure network connection security level.	1: digital signature 2: digital signature and encryption 3: digital signature, encryption, and user authentication 8: default value defined by backend system 9: maximum value that the current security product supports
jco.client.snc_myname	Indicates the secure network connection name. This property overrides the default secure network connection partner.	Token or identifier representing the external RFC program
jco.client.snc_lib	Identifies the path to the SAP crytpographic library that provides secure network connection service.	Full path and name of third-party security library. You must download and install the library from the SAP Service Marketplace.
jco.client.dest	Identifies a configured R/2 system defined in the sideinfo configuration.	
jco.client.dsr	Enables or disables jDSR monitoring.	0: off 1: on
jco.client.saplogon_id	Defines the string for SAPLOGON on 32-bit Windows.	String key to read parameters from the saplogon.ini file created by the SAPLogon GUI program on Windows
jco.client.extiddata	Provides data for external authentication (PAS). This is an old login mechanism similar to SSO; Sybase recommends that you do not use this approach.	

Name	Description	Supported values
jco.client.extidtype	Specifies type of external authentication (PAS). See External Authentication Data property.	

SAP DOE-C Properties

Configure SAP Data Orchestration Engine Connector (DOE-C) properties. This type of connection is available in the list of connection templates only when you deploy a DOE-C package. No template exists for these types of connections.

Note: If you change the username or password property of a DOE-C connection, you must reopen the same dialog and click `Test Connection` after saving. Otherwise the error state of this DOE-C package is not set properly, and an error message is displayed. This will not work if you click `Test Connection` before saving the properties.

Name	Description	Supported values
techuser-name	Specifies the SAP user account ID. The SAP user account is used during interaction between the connected SAP system and client for certain administrative activities, such as sending acknowledgment messages during day-to-day operations or "unsubscribe" messages if a subscription for this connection is removed. This account is not used for messages containing business data; those types of messages are always sent within the context of a session authenticated with credentials provided by the mobile client. The technical user name and password or certificateAlias must be set to perform actions on subscriptions. The certificateAlias is mutually exclusive with and overrides the technical user name and password fields if set. The technical user name and password fields can be empty, but only if `certificateAlias` is set.	Valid SAP login name for the DOE host system.

Name	Description	Supported values
techuser-password	Specifies the password for the SAP user account.	Valid password.
doe-soap-timeout	Specifies a timeout window during which unresponsive DOE requests are aborted.	Positive value (in seconds). The default is 420 (7 minutes).
doe-extract-window	Specifies the number of messages allowed in the DOE extract window.	Positive value (in messages). The default is 50. When the number of messages in the DOE extract window reaches 50% of this value, DOE-C sends a `StatusReqFromClient` message, to advise the SAP DOE system of the client's messaging status and acknowledge the server's state. The default value is 50.
doe-packetDropsize	Specifies the size, in bytes, of the largest JavaScript Object Notation (JSON) message that the DOE connector processes on behalf of a JSON client. The packet drop threshold size should be carefully chosen, so that it is larger than the largest message sent from the DOE to the client, but smaller than the maximum message size which may be processed by the client. Messages larger than the packet drop threshold size causes the subscription to enter the DOE packet drop state and become unusable.	Positive value (in bytes). The default is 1048576 bytes (1MB). Do not set higher than 2097152 (2MB), or lower than 4096.

Name	Description	Supported values
service-address	Specifies the DOE URL.	Valid DOE URL. If you are using DOE-C with SSO: • Modify the port from the standard http://host:8000 to https://host:8001/. • Add the certificate being used as the technical user and DOE-C end-point security profile certificate to the SAP DOE system's SSL Server certificate list by using the STRUST transaction. See your SAP documentation for details.
listener-url	Specifies the DOE-C server listener URL.	Valid DOE-C listener URL, for example http://host:8000/doe/publish
certificateAlias	Sets the alias for the Unwired Platform keystore entry that contains the X.509 certificate for Unwired Server's SSL peer identity. If you do not set a value, mutual authentication for SSL is not used when connecting to the Web service. If you are using DOE-C with SSO use the "SAP Technical User Certificate Alias" only for configurations which require the technical user to identify itself using an X.509 certificate; it specifies the Certificate Alias to be used as the technical user. This overrides the "Username" and "Password" settings normally used.	Valid certificate alias.

Name	Description	Supported values
login-required	Indicates whether authentication credentials are required to login. The default value is true. For upgraded packages, "login-required=false" gets converted to "login-required=true" and a No-Auth security configuration "DOECNoAuth" is assigned to the upgraded package.	A read-only property with a value of true.

Web Services Properties

Configure connection properties for the Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) architectures.

Name	Description	Supported values
password	Specifies the password for HTTP basic authentication, if applicable.	Password
address	Specifies a different URL than the port address indicated in the WSDL document at design time.	HTTP URL address of the Web service
user	Specifies the user name for HTTP basic authentication, if applicable.	User name
certificateAlias	Sets the alias for the Unwired Platform keystore entry that contains the X.509 certificate for Unwired Server's SSL peer identity. If you do not set a value, mutual authentication for SSL is not used when connecting to the Web service.	Use the alias of a certificate stored in the Unwired Server certificate keystore.

Proxy Endpoint Properties

Configure connection properties for the SAP Gateway proxy connection.

Name	Description	Supported values
password	Specifies the password for authentication.	Password

Name	Description	Supported values
address	URL address of the Gateway Proxy endpoint.	URL
user	Specifies the username for authentication.	User name
certificateAlias	Sets the alias for the Unwired Platform keystore entry that contains the X.509 certificate for Unwired Server's SSL peer identity. If you do not set a value, mutual authentication for SSL is not used when connecting to the Web service.	Use the alias of a certificate stored in the Unwired Server certificate keystore.
poolSize	An integer value representing the number of connections that can be made in the connection pool.	An integer.

Error Code Reference

Error codes are thrown with each `SUPAdminException`, to allow developers to diagnose what occurred when the exception is thrown. `${error_sub}` and `${reason_sub_x}` are placeholders for additonal information which will be provided at runtime.

Numeric Error Code	Message
00001	Failed to retrieve cluster properties (${error_sub}).
00002	Failed to authenticate the user (${error_sub}) as SUP administrator.
00003	Failed to validate the security configuration (${error_sub}).
00004	Failed to create the security configuration (${error_sub}).
00005	Cannot create the security provider (${error_sub}). The security configuration (${reason_sub}) is no longer valid or viable.
00006	Failed to delete the security configuration (${error_sub}).
00007	Cannot delete the selected security provider (${error_sub}). The security configuration (${reason_sub}) is no longer valid or viable.
00008	Failed to retrieve the selected security configuration (${error_sub}).
00009	Failed to retrieve the security configuration (${error_sub}) for the selected package. The package (${reason_sub_1}) is of the wrong type and therefore this operation (${reason_sub_2}) is not supported.
00010	Failed to update the selected security configuration (${error_sub}).
00011	Cannot delete the selected security provider (${error_sub}). The security configuration (${reason_sub}) is no longer valid or viable.
00012	Failed to create the authentication provider (${error_sub}). The authentication provider (${reason_sub}) cannot be located.
00013	Failed to retrieve the authentication provider (${error_sub}). The selected provider (${reason_sub}) does not exist.
00014	Failed to retrieve the authentication provider (${error_sub}). The authentication provider (${reason_sub}) cannot be located.
00015	Failed to create the authorization provider (${error_sub}). The authorization provider (${reason_sub}) cannot be located.

Numeric Error Code	Message
00016	Failed to retrieve the authorization provider (${error_sub}). The selected provider (${reason_sub}) does not exist.
00017	Failed to retrieve the authorization provider (${error_sub}). The authorization provider (${reason_sub}) cannot be located.
00018	Failed to created the attribution provider (${error_sub}). The attribution provider (${reason_sub}) cannot be located.
00019	Failed to retrieve the attribution provider (${error_sub}). The selected provider (${reason_sub}) does not exist.
00020	Failed to retrieve the attribution provider (${error_sub}). The attribution provider (${reason_sub}) cannot be located.
00021	Failed to create the audit provider (${error_sub}). The audit provider (${reason_sub}) cannot be located.
00022	Failed to retrieve the audit provider (${error_sub}). The selected provider (${reason_sub}) does not exist.
00023	Failed to create the audit destination (${error_sub}). The audit provider (${reason_sub}) cannot be located.
00024	Failed to retrieve the audit destination (${error_sub}).
00025	Failed to create the audit filter (${error_sub}). The audit provider (${reason_sub}) cannot be located.
00026	Failed to retrieve the audit filter (${error_sub}). The audit provider (${reason_sub}) cannot be located.
00027	Failed to create the audit formatter (${error_sub}). The audit provider (${reason_sub}) cannot be located.
00028	Failed to retrieve the audit formatter (${error_sub}). The audit provider (${reason_sub}) cannot be located.
00029	Cannot delete the selected security provider (${error_sub}). This security provider (${reason_sub}) does not exist.
00030	Cannot update the selected security provider (${error_sub}). This security provider (${reason_sub}) does not exist.
00031	Failed to enable the domain (${error_sub}).

Numeric Error Code	Message
00032	Failed to create the domain (${error_sub}).
00033	Failed to delete the domain (${error_sub}).
00034	Failed to retrieve the domain (${error_sub}).
00035	Failed to update the domain (${error_sub}) properties.
00036	Failed to retrieve the domain log configuration (${error_sub}).
00037	Failed to retrieve the domain log (${error_sub}). A configuration property (${reason_sub}) is not supported.
00038	Cannot retrieve the domain log configuration (${error_sub}).
00039	Failed to update the domain log configuration (${error_sub}).
00040	Failed to retrieve the domain log purge time threshold value (${error_sub}).
00041	Failed to update the domain log purge time threshold value (${error_sub}).
00042	Package deployment failed (${error_sub}).
00043	Failed to deploy selected package (${error_sub}). You must select a security configuration.
00044	Failed to deploy the selected package (${error_sub}). The package (${reason_sub}) is the wrong type and this operation is not supported.
00045	Failed to deploy package (${error_sub}). Either the deployment unit (${reason_sub_1}) does not exist, or the file (${reason_sub_2}) may be invalid or corrupted.
00046	Failed to deploy package (${error_sub}). The deployment unit may be invalid or corrupted.
00047	Failed to deploy package (${error_sub}). The deployment descriptor may be invalid or corrupted.
00048	Failed to deploy package (${error_sub}). A required property (${reason_sub}) has not been configured.
00049	Failed to deploy package (${error_sub}). A required property (${reason_sub}) has not been configured.
00050	Package export failed (${error_sub}).
00051	Failed to export the selected package (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.

Numeric Error Code	Message
00052	Failed to export package (${error_sub}). The file (${reason_sub}) does not exist.
00053	Package import failed (${error_sub}).
00054	Failed to enable package (${error_sub}).
00055	Failed to enable the selected package (${error_sub}). The package (${reason_sub}) is the wrong type and this operation (${reason_sub}) is not supported.
00056	Failed to delete the selected package (${error_sub}).
00057	Failed to retrieve package(s).
00058	Failed to retrieve cache group(s) (${error_sub}).
00059	Failed to retrieve the cache group (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00060	Failed to update cache group(s) (${error_sub}).
00061	Failed to update the cache group (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00062	Failed to retrieve the cache group schedule (${error_sub}).
00063	Failed to retrieve the cache group schedule (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00064	Failed to update the cache group schedule (${error_sub}).
00065	Failed to update the cache group schedule (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00066	Failed to retrieve the personalization key (${error_sub}).
00067	Failed to retrieve the personalization key (${error_sub}). The package (${reason_sub_2}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00068	Failed to retrieve the package role mapping (${error_sub}).
00069	Failed to retrieve the package role mappings (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00070	Failed to updated the package role mapping (${error_sub}).
00071	Failed to update the package role mapping (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.

Numeric Error Code	Message
00072	Failed to retrieve the synchronization group (${error_sub}).
00073	Failed to retrieve the synchronization group (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00074	Failed to update the synchronization group (${error_sub}).
00075	Failed to update the synchronization group (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00076	Failed to retrieve the MBO(s).
00077	Failed to retrieve the MBO(s). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00078	Failed to retrieve the configured MBO server connection (${error_sub}).
00079	Failed to delete error history of the MBO (${error_sub}).
00080	Failed to retrieve the error history of the MBO (${error_sub}).
00081	Failed to retrieve the last valid playback timestamp for the MBO (${error_sub}).
00082	Failed to retrieve the operation(s) (${error_sub}).
00083	Failed to retrieve the configured server connection of the operation (${error_sub}).
00084	Failed to delete the error history of the operation (${error_sub}).
00085	Failed to retrieve the error history of the operation (${error_sub}).
00086	Failed to retrieve the last valid playback timestamp for the operation (${error_sub}).
00087	Failed to retrieve package log configuration (${error_sub}).
00088	Failed to update package log configuration (${error_sub}).
00089	Failed to enable package synchronization tracing (${error_sub}).
00090	Failed to enable package synchronization tracing (${error_sub}). The package (${reason_sub_1}) is the wrong type and this operation (${reason_sub_2}) is not supported.
00091	Failed to retrieve the package log level (${error_sub}).
00092	Failed to update the package log level (${error_sub}).
00093	Failed to ping the replication package subscription (${error_sub}).

Numeric Error Code	Message
00094	Failed to ping the replication package subscription (${error_sub}). The selected subscription (${reason_sub}) does not exist.
00095	Failed to delete the replication package subscription (${error_sub}).
00096	Failed to delete the replication package subscription(s) (${error_sub}). The selected subscription(s) (${reason_sub}) does(do) not exist.
00097	Failed to retrieve the replication package subscription(s) (${error_sub}).
00098	Failed to update the replication package subscription (${error_sub}).
00099	Failed to create the replication package subscription template (${error_sub}).
00100	Failed to delete the replication package subscription template(s) (${error_sub}).
00101	Failed to retrieve the replication package subscription template(s) (${error_sub}).
00102	Failed to suspend the messaging package subscription(s) (${error_sub}).
00103	Failed to resume the messaging package subscription(s) (${error_sub}).
00104	Failed to delete the messaging package subscription(s) (${error_sub}).
00105	Failed to retrieve the messaging package subscription(s) (${error_sub}).
00106	Failed to reset the messaging package subscription(s) (${error_sub}).
00107	Failed to re-synchronize the DOEC package subscription(s) (${error_sub}).
00108	Failed to reset the DOEC package subscription(s) (${error_sub}).
00109	Failed to delete the DOEC package subscription(s) (${error_sub}).
00110	Failed to retrieve the DOEC package subscription(s) (${error_sub}).
00111	Failed to update the DOEC package subscription(s) (${error_sub}).
00112	Failed to reset the DOEC package subscription(s) (${error_sub}).
00113	Failed to retrieve the log level for DOEC package subscription (${error_sub}). The package (${reason_sub_1}) is of the wrong type and therefore this operation (${reason_sub_2}) is not supported.
00114	Failed to update the log level for DOEC package subscription(s) (${error_sub}).
00115	Failed to retrieve the log level for DOEC package subscription(s) (${error_sub}). The package (${reason_sub_1}) is of the wrong type and therefore this operation (${reason_sub_2}) is not supported.

Numeric Error Code	Message
00116	Failed to connect to the configured server connection (${error_sub}).
00117	Failed to create the server connection (${error_sub}).
00118	Failed to create the server connection (${error_sub}). The server connection (${reason_sub_1}) is of the wrong type and therefore this operation (${reason_sub_2}) is not supported.
00119	Failed to delete the server connection (${error_sub}).
00120	Failed to delete the server connection (${error_sub}). The server connection (${reason_sub_1}) is of the wrong type and therefore this operation (${reason_sub_2}) is not supported.
00121	Failed to retrieve the server connection(s) (${error_sub}).
00122	Failed to update the server connection (${error_sub}).
00123	Failed to retrieve the domain-level role mapping (${error_sub}).
00124	Failed to update the domain-level role mapping (${error_sub}).
00125	Failed to create the domain administrator (${error_sub}).
00126	Failed to delete the domain administrator (${error_sub}).
00127	Failed to retrieve the domain administrator(s) (${error_sub}).
00128	Failed to update the domain administrator (${error_sub}).
00129	Failed to authenticate the user as SUP domain administrator (${error_sub}).
00130	Failed to create the monitoring profile (${error_sub}).
00131	Failed to delete the monitoring profile (${error_sub}).
00132	Failed to retrieve the monitoring profile(s) (${error_sub}).
00133	Failed to update the monitoring profile (${error_sub}).
00134	Failed to update the monitoring profile (${error_sub}). A property (${reason_sub}) uses an incorrect value.
00135	Failed to export monitoring data (${error_sub}).
00136	Failed to delete monitoring data (${error_sub}).
00137	Failed to retrieve monitoring data (${error_sub}). A required parameter (${reason_sub}) is missing.

Numeric Error Code	Message
00138	Failed to retrieve monitoring data (${error_sub}). A required parameter (${reason_sub}) is not expected.
00139	Failed to retrieve monitoring data (${error_sub}). A required parameter (${reason_sub}) is empty
00140	Failed to retrieve the replication package monitoring data (${error_sub}).
00141	Failed to retrieve the replication package history (${error_sub}).
00142	Failed to retrieve the replication package performance (${error_sub}).
00143	Failed to retrieve the messaging package monitoring data (${error_sub}).
00144	Failed to retrieve the messaging package history (${error_sub}).
00145	Failed to retrieve the messaging package performance data (${error_sub}).
00146	Failed to retrieve the messaging queue statistics (${error_sub}).
00147	Failed to retrieve the data change notfication history data (${error_sub}).
00148	Failed to retrieve the data change notification performance data (${error_sub}).
00149	Failed to retrieve the package statistics (${error_sub}).
00150	Failed to retrieve the operation statistics (${error_sub}).
00151	Failed to retrieve user access history (${error_sub}).
00152	Failed to retrieve the package-level cache group performance data (${error_sub}).
00153	Failed to retrieve the package-level cache group statistics (${error_sub}).
00154	Failed to retrieve MBO-level cache group statistics (${error_sub}).
00155	Failed to start Unwired Server (${error_sub}). The path (${reason_sub}) to the server does not exist.
00156	Failed to start Unwired Server (${error_sub}). Sybase Control Center is not installed on the same host computer, and this operation (${reason_sub}) cannot be performed remotely.
00157	Failed to connect to Unwired Server (${error_sub}).
00158	Failed to stop Unwired Server (${error_sub}).
00159	Failed to stop Unwired Server (${error_sub}). The path (${reason_sub}) to the server does not exist.

Numeric Error Code	Message
00160	Failed to stop Unwired Server (${error_sub}). Sybase Control Center is not installed on the same host computer, and this operation (${reason_sub}) cannot be performed remotely.
00161	Failed to restart Unwired Server (${error_sub}).
00162	Failed to restart Unwired Server (${error_sub}). The path (${reason_sub}) to the server does not exist.
00163	Failed to restart Unwired Server (${error_sub}). Sybase Control Center is not installed on the same host computer, and this operation (${reason_sub}) cannot be performed remotely.
00164	Failed to suspend Unwired Server (${error_sub}).
00165	Failed to resume Unwired Server (${error_sub}).
00166	Failed to retrieve Unwired Server properties (${error_sub}).
00167	Failed to create the Unwired Server configuration (${error_sub}). The specified configuration type (${reason_sub}) does not exist.
00168	Failed to create the Unwired Server configuration (${error_sub}). A parameter (${reason_sub}) is not expected.
00169	Failed to delete the Unwired Server configuration (${error_sub}). The specified configuration (${reason_sub}) does not exist.
00170	Failed to update the Unwired Server configuration (${error_sub}).
00171	Failed to update the Unwired Server configuration (${error_sub}).The selected Unwired Server (${reason_sub}) does not exist.
00172	Failed to update the Unwired Server configuration (${error_sub}). A property value (${reason_sub}) in the configuration is not supported.
00173	Failed to initialize the administration listener (${error_sub}). The listener (${reason_sub}) has not been configured.
00174	Failed to secure the administration listener (${error_sub}). The listener (${reason_sub}) has not been configured.
00175	Failed to retrieve the key store configuration (${error_sub}). The key store (${reason_sub}) has not been configured.

Numeric Error Code	Message
00176	Failed to retrieve the key store configuration (${error_sub}). The key store configuration (${reason_sub}) is corrupted.
00177	Failed to retrieve the trust store configuration (${error_sub}). The trust store (${reason_sub}) is not configured.
00178	Failed to retrieve the trust store configuration (${error_sub}). The key store configuration (${reason_sub}) is corrupted.
00179	Failed to retrieve the cache database configuration (${error_sub}). The cache database (${reason_sub}) has not been configured.
00180	Failed to retrieve the replication synchronization server configuration (${error_sub}). The synchronization server (${reason_sub}) is not configured.
00181	Failed to retrieve the messaging synchronization server configuration (${error_sub}). The synchronization server (${reason_sub}) is not configured.
00182	Failed to retrieve the replication push notification configuration (${error_sub}). The replication push component (${reason_sub}) is not configured.
00183	Failed to retrieve the replication push notification gateway configuration (${error_sub}). The gateway (${reason_sub}) is not available.
00184	Failed to validate the Unwired Server log configuration (${error_sub}).
00185	Failed to retrieve the Unwired Server log configuration (${error_sub}).
00186	Failed to update the Unwired Server log configuration (${error_sub}).
00187	Failed to create the Unwired Server log appender (${error_sub}). The log appender type is not installed.
00188	Failed to create the Unwired Server log appender (${error_sub}). The log bucket type (${reason_sub}) is not installed.
00189	Failed to delete the Unwired Server log appender (${error_sub}). The log appender (${reason_sub}) does not exist.
00190	Failed to update the Unwired Server log appender (${error_sub}). The log appender (${reason_sub}) does not exist.
00191	Failed to update the Unwired Server log appender (${error_sub}). The log bucket type (${reason_sub}) is not installed.

Numeric Error Code	Message
00192	Failed to create the Unwired Server log bucket (${error_sub}). The log appender (${reason_sub}) does not exist.
00193	Failed to create the Unwired Server log file (${error_sub}). The log appender type is not installed.
00194	Failed to delete the Unwired Server log bucket (${error_sub}). The log bucket (${reason_sub}) does not exist.
00195	Failed to update the Unwired Server log bucket (${error_sub}). The log bucket (${reason_sub}) does not exist.
00196	Failed to delete the Unwired Server log (${error_sub}).
00197	Failed to retrieve the Unwired Server log (${error_sub}).
00198	Failed to create the Unwired Server log filter (${error_sub}).
00199	Failed to retrieve the Unwired Server log filter (${error_sub}).
00200	Failed to connect to the Sybase Control Center (${error_sub}).
00201	Failed to borrow a connection from the connection pool of Sybase Control Center (${error_sub}).
00202	Failed to return a connection to the connection pool of Sybase Control Center (${error_sub}).
00203	Cannot retrieve the Unwired Server location (${error_sub}). The login has not authenticated.
00204	Cannot retrieve the Unwired Server location (${error_sub}). The login is not authorized to access this server.
00205	Cannot retrieve the Unwired Server location (${error_sub}). The Sybase Control Center connection has failed.
00206	Failed to load the file (${error_sub}). The file does not exist.
00207	Failed to load the the administration interface (${error_sub}).
00208	Failed to initialize the administration interface (${error_sub}).
00209	Failed to retreive data for administration interface (${error_sub}).
00210	Failed to validate the property (${error_sub}). A value other than NULL is required.
00211	Failed to validate the property (${error_sub}). A monitored target is required.

Numeric Error Code	Message
00212	The value entered exceeds the maximum value allowed (${error_sub}).
00213	The value entered exceeds the minimum value allowed (${error_sub}).
00214	Failed to invoke method (${error_sub}). The parameter (${reason_sub}) is either the wrong type or uses the wrong value.
00215	Failed to invoke method (${error_sub}). The parameter (${reason_sub}) requires a value.
00216	Failed to invoke method (${error_sub}). The method (${reason_sub}) is not implemented.
00217	Failed to invoke method (${error_sub}). Access to the method (${reason_sub}) is denied.
00218	Monitoring data retrieve failed.
00219	Messaging-based synchronization server workflow (${error_sub}) retrieve failed.
00220	Messaging-based synchronization server workflow error (${error_sub}) delete failed.
00221	Java virtual machine start up options retrieve failed because java virtual machine start up options does not exist.
00222	Object (${error_sub}) create failed because parameter (${reason_sub}) not supported.
00223	Messaging-based synchronization server apple push notification certificate (${error_sub}) list failed.
00224	Device (${error_sub}) list failed.
00225	Messaging-based synchronization server email (${error_sub}) retrieve failed.
00226	Device (${error_sub}) retrieve failed.
00227	SSL Security profile (${error_sub}) delete failed because parameter (${reason_sub}) null value not allowed.
00228	Messaging-based synchronization server template (${error_sub}) list failed.
00229	Secure administration listener (${error_sub}) update failed because parameter (${reason_sub}) null value not allowed.
00230	Messaging-based synchronization server workflow context variable (${error_sub}) update failed.
00231	Messaging-based synchronization server template (${error_sub}) retrieve failed.

Numeric Error Code	Message
00232	User (${error_sub}) delete failed.
00233	Replication notification gateway update failed because parameter (${reason_sub}) null value not allowed.
00234	HTTP listener(s) (${error_sub}) delete failed because parameter (${reason_sub}) null value not allowed.
00235	SSL Security profile (${error_sub}) update failed because parameter (${reason_sub}) null value not allowed.
00236	Replication notification gateway enbale failed because parameter (${reason_sub}) null value not allowed.
00237	Messaging-based synchronization server workflow error (${error_sub}) list failed.
00238	Messaging-based synchronization server workflow (${error_sub}) update failed.
00239	Messaging-based synchronization server workflow (${error_sub}) delete failed.
00240	Java virtual machine start up options retrieve failed because java virtual machine start up options corrupted.
00241	HTTP listener(s) (${error_sub}) update failed because parameter (${reason_sub}) null value not allowed.
00242	Messaging-based synchronization server workflow (${error_sub}) create failed.
00243	Messaging-based synchronization server (${error_sub}) list failed.
00244	Messaging-based synchronization server apple push notification certificate (${error_sub}) update failed.
00245	Messaging-based synchronization server email (${error_sub}) update failed.
00246	Messaging-based synchronization server apple push notification certificate (${error_sub}) delete failed.
00247	Device (${error_sub}) update failed.
00248	Device (${error_sub}) delete failed.
00249	Secure HTTP listener(s) (${error_sub}) delete failed because parameter (${reason_sub}) null value not allowed.
00250	Messaging-based synchronization server workflow display name (${error_sub}) update failed.

Numeric Error Code	Message
00251	Messaging-based synchronization server apple push notification certificate (${error_sub}) create failed.
00252	Messaging-based synchronization server email (${error_sub}) create enabled.
00253	Device (${error_sub}) create failed.
00254	Monitored object (${error_sub}) update failed because parameter (${reason_sub}) invalid.
00255	License retrieve failed.
00256	Monitored object (${error_sub}) update failed because parameter (${reason_sub}) null value not allowed.
00257	Object (${error_sub}) create failed because parameter (${reason_sub}) null value not allowed.
00258	Messaging-based synchronization server workflow context variable (${error_sub}) list failed.
00259	Messaging-based synchronization server template (${error_sub}) delete failed.
00260	User (${error_sub}) list failed.
00261	Secure HTTP listener(s) (${error_sub}) update failed because parameter (${reason_sub}) null value not allowed.
00262	Messaging-based synchronization server email (${error_sub}) validate failed.
00263	Administration listener update failed because parameter (${reason_sub}) null value not allowed.
00264	Replication notifier retrieve failed because parameter (${reason_sub}) null value not allowed.
00265	Messaging-based synchronization server workflow (${error_sub}) list failed.
00266	Failed to start Unwired Server (${error_sub}). Server command and control (${reason_sub}) is in progress.
00267	Failed to stop Unwired Server (${error_sub}). Server command and control (${reason_sub}) is in progress.
00268	Failed to restart Unwired Server (${error_sub}). Server command and control (${reason_sub}) is in progress.

Numeric Error Code	Message
00269	Cannot deploy the package as (${reason_sub}) type. When deployment unit contains an Online Cache Group, the package must be deployed as a MESSAGING package.
00270	Failed to delete the selected package (${error_sub}) because of invalid SAP credentials.
00271	Failed to delete the Mobile Workflow (${error_sub}) because of invalid SAP credentials.
00272	Failed to delete the DOEC package subscription(s) (${error_sub}) because of invalid SAP credentials.
00273	Failed to re-synchronize the DOEC package subscription(s) (${error_sub}) because of invalid SAP credentials.
00274	Failed to enable the scheduled purge task (${error_sub}).
00275	Failed to retrieve the scheduled purge task (${error_sub}) enable status.
00276	Failed to purge the synchronization cache.
00277	Failed to purge the online cache.
00278	Failed to purge the client log.
00279	Failed to purge the error history.
00280	Failed to purge the subscription.
00281	Failed to retrieve the purge option (${error_sub}).
00282	Failed to set the purge option (${error_sub}).
00283	Failed to retrieve the purge task (${error_sub}) schedule.
00284	Failed to update the purge task (${error_sub}) schedule.
00285	Failed to purge the package's synchronization cache.
00286	Failed to purge the package's online cache.
00287	Failed to purge the package's client log.
00288	Failed to purge the package's error history.
00289	Failed to purge the package's subscription.
00290	Failed to purge devices.
00291	Failed to purge users.

Numeric Error Code	Message
00292	Failed to replace mobile workflow (${error_sub}) certificate.
00293	Failed to replace mobile workflow certificate. The mobile workflow does not support certificate based authentication.
00294	Messaging-based synchronization server workflow context variable (${error_sub}) update failed. The parameter is invalid.

Backward Compatibility

When upgrading from a previous version of Sybase Unwired Platform, certain APIs are no longer supported, or are supported with limitations.

These APIs are no longer supported:

- `SUPDeviceUser`: all methods of this class throw an UnsupportedOperationException when called.
- `SUPDomainLog`: all methods of this class except the `getContext()` method throw an UnsupportedOperationException when called.
- `SUPSecurityConfiguration`: the following methods of this class throw exceptions when called, because the attribution provider configuration is no longer exposed.
  - getInstalledAttributionProviders
  - getActiveAttributionProviders
  - getActiveAttributionProvider
  - addActiveAttributionProvider
  - deleteActiveAttributionProvider
  - updateActiveAttributionProvider
  - moveUpActiveAttributionProvider
  - moveDownActiveAttributionProvider

These APIs are supported with limitations:

Table 234. SUPDomain

Method	Reason	Notes
Deploy	Package unification	You can still deploy the deployment units of the previous version. However, the package type passed to this method is ignored. Clients of the previous version see the newly deployed package as an RBS package, however, the Unwired Server treats it as a unified package. Packages deployed prior to the upgrade retain their type information for the older version of the client.

Table 235. SUPPackage

Method	Reason	Notes
getProperties()	Package unification	Clients of the previous version see the newly deployed package as an RBS package, however, the Unwired Server treats it as a unified package. Packages deployed prior to the upgrade retain their type information for the older version of the client.

Note: If using the 2.0 version of the Management API client to connect to a Sybase Unwired Platform 2.1 installation, you must get the uaf-client.jar shipped with Sybase Unwired Platform 2.0.1 in the Management API client libraries folder.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

T

U