SYBASE®
An SAP® Company

Developer Guide: Unwired Server Runtime

Sybase Unwired Platform 2.2 SP02

DOCUMENT ID: DC01852-01-0222-01
LAST REVISED: January 2013

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

Contents

Security API

Use the Security API to develop a custom authentication or authorization provider. The audience is advanced developers who are experienced in working with APIs, but who may be new to Sybase® Unwired Platform.

This guide describes the Common Security Infrastructure (CSI) component, the security framework in Sybase Unwired Platform, and how to develop a custom provider for extending the authentication and authorization functionality to use a back-end repository that is not supported by the default providers included in Sybase Unwired Platform.

Introducing Security API

Use the Security API to develop a custom provider. The audience is advanced developers who are experienced in working with APIs, but who may be new to Sybase Unwired Platform.

This section describes the features and usage of the Security API, the Sybase Unwired Platform security framework, as well as configuration, localization, internationalization, logging, and error handling related to CSI.

Security API Features

Sybase Unwired Platform includes a Security API that enables users to develop a custom authentication or authorization provider.

Sybase Unwired Platform delegates the functions of storing and maintaining users and access control rules to the enterprise's existing security solutions. It uses a plug-in model to delegate the security checks to the configured providers using the CSI component.

CSI has a service provider plug-in model that integrates with the customer's existing security infrastructure. If none of the default providers shipped with Sybase Unwired Platform meet the security needs, you can use the Security API to implement a custom login module, authorizer, or attributer that interfaces with a security back-end of your choice and plug it into Sybase Unwired Platform.

Companion Documentation for Security API

Companion guides include:

- *Security*
- *Sybase Control Center for Sybase Unwired Platform*
- *Troubleshooting*

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

Documentation Roadmap for Sybase Unwired Platform

Sybase Unwired Platform documents are available for administrative and mobile development user roles. Some administrative documents are also used in the development and test environment; some documents are used by all users.

See *Documentation Roadmap* in *Fundamentals* for document descriptions by user role.

Check the Sybase® Product Documentation Web site regularly for updates: *http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories* , then navigate to the most current version.

Quick Start Task Flow

This section provides a quick reference to information and task flows relevant to Security API.

1. *Getting Started with CSI Provider Development*

   This section describes the task flow for developing a custom CSI provider.
2. *Developing a Custom Provider*

   Develop your custom provider by writing a provider metadata file specifying valid configuration options.
3. *Packaging a Custom Provider into a JAR File*

   After you have developed a custom provider, you must package the provider for installation.
4. *Installing a Custom Provider JAR File on Unwired Server*

   After you have developed your custom provider and packaged the JAR file, you must install the JAR file.
5. *Validating a Custom Provider Using the HTTP Channel Debug Application*

   After installing your custom provider, validate your configuration using the HTTP Channel debug application.

Getting Started with CSI Provider Development

This section describes the task flow for developing a custom CSI provider.

Before you get started with CSI provider development, be aware of provider project dependencies including:

- `csi-core.jar` - This JAR encapsulates the CSI security framework packages, containing classes and interfaces that are required for developing CSI providers. More details on the packages contained in this JAR file can be found in *Developing a Custom Provider* and in the *CSI SDK API Reference*.

The file can be found at the following locations:
- *SUP_HOME*\Servers\UnwiredServer\lib\ext\
- *SUP_HOME*\Servers\UnwiredServer\ServerSDK\securityAPI\lib

- providermetadata44.xsd - This file specifies the XML schema that dictates the structure of a provider metadata file describing the provider configuration options, type, and default values. The provider metadata file is used to display the configuration options for the provider in Sybase Control Center and should be present in every CSI provider JAR. For more details on the provider metadata file, see *Developing a Custom Provider* and *Packaging a Custom Provider into a JAR File*. Also refer to *Validating a Custom Provider Using the HTTP Channel Debug Application*.

  This file can be accessed from one of the following locations:
  - *SUP_HOME*\Servers\UnwiredServer\ServerSDK\securityAPI\metadata\
  - From within the csi-core-admin.jar at com\sybase\security\admin\ (the csi-core-admin.jar file can be found at *SUP_HOME*\Servers\UnwiredServer\lib\ext)

A CSI sample provider project, complete with working code and supporting documentation, can be found at: *SUP_HOME*\Servers\UnwiredServer\ServerSDK\securityAPI\samples. This sample project provides a hands-on approach to exploring the CSI SDK.

The quickStartGuide.html file in the CSI SDK describes the steps to set up the sample provider project. This file can be found at: *SUP_HOME*\Servers\UnwiredServer\ServerSDK\securityAPI\.

Developing a Custom Provider

Develop your custom provider by writing a provider metadata file specifying valid configuration options.

Packages in csi-core.jar library

Before starting development, be sure to understand the packages in the CSI core library. Packages include:

- The com.sybase.security package contains the main classes and interfaces that define the Sybase CSI framework.
- The com.sybase.security.authorization package provides various types of security authorization request types with which a CSI consumer could construct more complex authorization requests.
- The com.sybase.security.callback package contains the callback and callback handler implementations used and supported by the default authentication providers.

- The `com.sybase.security.core` package contains the default provider implementations packaged as part of `csi-core.jar` file, as well as utility classes that are useful in implementing new providers.
- The `com.sybase.security.provider` package contains the interfaces implemented by various provider types and helper abstract classes. These helper classes provide an abstract implementation of the corresponding provider interfaces, complete with placeholders for all of the implemented interface's methods, so that subclasses need only to override relevant methods. The AbstractLoginModule, AbstractAuthorizer, and AbstractAttributer present in this package can be extended to overwrite the necessary methods to develop a custom login module, authorizer, and attributer respectively.

For more information, see *Security API Reference*.

Writing a provider metadata file

The CSI provider metadata XML file specifies the valid configuration options, including names, types, default values, as well as required or optional properties, for a particular CSI provider. The custom provider's details and configuration options in Sybase Control Center is derived from the contents of this file. This metadata file should be named `sybcsi-provider.xml` and should conform to the latest `providermetadata.xsd` file included in the Security API.

A sample `sybcsi-provider.xml` file is available in the sample project located at *SUP_HOME*`\Servers\UnwiredServer\ServerSDK\securityAPI\samples\src\main\resources`.

Packaging a Custom Provider into a JAR File

After you have developed a custom provider, you must package the provider for installation.

The custom CSI provider classes are packaged into a JAR file that adheres to the standard JAR file specification at *http://docs.oracle.com/javase/6/docs/technotes/guides/jar/jar.html*.

In addition to the standard layout, include a provider metadata file named `sybcsi-provider.xml` at the root directory level of the provider JAR. The details about the custom provider and its configuration options displayed in Sybase Control Center come from this file.

Installing a Custom Provider JAR File on Unwired Server

After you have developed your custom provider and packaged the JAR file, you must install the JAR file.

You must make your new CSI provider available to Unwired Server so it can be used in your security configuration. The JAR file needs to be installed onto the server instance by following the steps below:

1. Shutdown the Sybase Unwired Platform server.

2. Copy the custom provider JAR file into the *SUP_HOME*\Servers\UnwiredServer\lib\ext folder.
3. Create an empty file named sup.cff in the *SUP_HOME*\Servers\UnwiredServer\bin\private directory.
4. Start the server using the batch command start mode in order to regenerate the class path.

To confirm that the installation succeeded, create a security configuration using Sybase Control Center, and confirm the provider classes packaged in the installed JAR are listed as available providers. For more information, see *Security Configurations* in *Sybase Control Center for Sybase Unwired Platform*.

Validating a Custom Provider Using the HTTP Channel Debug Application

After installing your custom provider, validate your configuration using the HTTP Channel debug application.

To validate that a security configuration in Unwired Server is setup correctly, use the HTTP Channel debug Web application pre-installed in the Sybase Unwired Platform server.

For more information, see *Debug Security Configurations Using HTTP Channel* in the *Troubleshooting*.

The Sybase Unwired Platform Security Framework

Unwired Platform delegates the functions of storing and maintaining users and access control rules to the enterprise's existing security solutions. It uses a plug-in model to delegate the security checks to the configured providers that uses the CSI component.

CSI has a service provider plug-in model that integrates with the customer's existing security infrastructure. If none of the default providers shipped with Sybase Unwired Platform meet your security needs, you can implement a custom login module, authorizer, and attributer that interfaces with the security back-end of choice, and plug it into Sybase Unwired Platform as long as it implements the interfaces described in this section.

Administrative access to the Sybase Unwired Platform server and the domains is controlled by the providers in the "admin" security configuration. Access to the packages deployed in the various Sybase Unwired Platform domains is controlled by the security configuration associated with the packages. Different message-based synchronization (MBS) subscriptions and the mobile business object (MBO) operations in a package may require different roles to access them. The user authentication and role requirement at runtime is enforced by the security configuration associated with the package. The security configuration includes authentication, authorization, attribution, and audit providers.

A CSI SecContextFactory instance is associated with a single security configuration in Sybase Unwired Platform and is cached to avoid the overhead in creating and initializing the factory for each authentication. A separate SecContext instance is created for each client

authentication request, and is saved for the duration of the client connection. For example, a replication-based synchronization (RBS) session involves authenticating the user, the begin sync event, and the end sync event. For MBS, each message sent by the client results in a separate session in Sybase Unwired Platform that triggers user authentication, and an authorization check to verify the user's authority to execute the operation and method invocation requested in the message.

Authentication

Authentication within CSI is performed internally using the Java Authentication and Authorization Services (JAAS) model by supplying the client credentials in an object instance that implements the `javax.security.auth.callback.CallbackHandler` interface.

Sybase Unwired Platform caches a successful authentication result for a configurable time interval. If another authentication request using the same credentials is received within the authentication cache time interval after a successful authentication, the request is not delegated to CSI.

The authentication provider interfaces in CSI are primarily based on JAAS. The goal of the design is to allow any implementation of the JAAS pluggable authentication module system to alternatively plug into the CSI framework. Specifically, the `javax.security.auth.spi.LoginModule` interface must be implemented by all authentication providers. CSI provides a flexible mechanism for defining active authentication providers and their control flags. All control flags defined in JAAS are fully implemented in the Sybase Unwired Platform architecture. See the JAAS documentation for a complete discussion of the configuration options available with JAAS login modules.

CSI instantiates new authentication provider instances, upon demand, for each authentication request. This is distinctly different from authorization and attribution providers, for which instances are reused and must be thread-safe. Normally, the first method called on a security context after creation results in authentication. The framework creates the authentication providers using the default public, no-argument constructor. If a provider class does not have a constructor meeting these requirements, a `com.sybase.security.SecException` object is created and thrown with the appropriate detail information.

After initialization, the provider methods are called in the following order; however, not all methods are necessarily called:

1. `initialize()`
2. `login()`
3. `commit()`

At any time after the `login()` method is called on a provider, be prepared for the abort method to be called to abort the authentication attempt.

The `initialize()` method call includes four arguments. The callback handler is how an authentication provider should attempt to retrieve credentials from the client. Also included is

a `javax.security.auth.Subject` object, into which an authentication provider typically adds custom principals and credentials. The next argument is a `java.util.Map` instance that provides a way for all configured providers to share information. There are no standard way for providers to communicate with each other, but many of the providers that ship with the JDK support certain well-defined communication capabilities. The last argument is also a `java.util.Map` instance, which represents all of the provider-specific configuration options. Included in this argument are any URLs, user names, passwords, and anything else a provider needs to know about its environment to provide authentication services.

Principals play a crucial role in the JAAS and Java security models. It is through principals that a direct JAAS client can discover an authenticated user. A typical authentication provider might add one or more principals to the subject's principal set (see the `Subject.getPrincipals()` method) after verifying the user's identity. Each principal instance must include at least one method, `getName()`, which is defined in the `java.security.Principal` interface. The return value of this method is a unique string that represents this user in the underlying security system.

For an example, the LDAPLoginModule adds these principal objects to the Subject after authentication

- `com.sybase.security.ldap.LDAPDNPrincipal`, which represents the user's LDAP DN
- `com.sybase.security.ldap.LDAPUsernamePrincipal`, which represents the name used to authenticate the user

Either of these objects can be used individually to determine exactly who the authenticated user is, which makes them good candidates for principals. Examples of other potentially good choices for principals include e-mail address, phone number, or user name. Poor choices for principals include first name, last name, or birthday.

Principals themselves are not useful without any context about where they came from and what format they are in. After authentication is complete and the user's principals are collected, CSI makes a pass through the set of principals and attempts to classify them according to some rudimentary rules. In particular, two categories are defined: name and ID. A principal that is the name of a user is thought to be a principal whose string representation presents itself in a relatively human-readable form. Something like the user name or e-mail address might be considered a name principal. A principal that is an ID principal is typically not very human-friendly–it might be a large number or a string of seemingly random characters that form a machine-readable unique identifier.

Both of these principals are useful in their own places, so CSI provides a way to categorize a principal as a name and/or ID, or neither. You can use the empty interfaces `com.sybase.security.providers.SecIDPrincipal` and `com.sybase.security.providers.SecNamePrincipal` to tag custom authentication providers' principals as name or ID, or both. If a principal instance implements the `SecIDPrincipal` interface, then the value returned from the `getName()` method is

added to the security context as an ID attribute. Logically, principals that implement the `SecNamePrincipal` interface are added as NAME attributes in the security context. The actual principal instances themselves remain available to CSI clients and providers within the JAAS subject object, which you can retrieve using the `SecSubject.getJAASSubject()` method.

To facilitate role-based authentication, login modules may also retrieve the roles assigned to a user and add them as Principals to the authenticated subject. If these principals implement the `interface com.sybase.security.core.RoleCheck`, you can use the default authorizer `com.sybase.security.core.RoleCheckAuthorizer`, which is part of the Sybase Unwired Platform configurations, to perform authorization checks without the need for an authorization provider for the specific back end.

Table 1. Control Flag Settings for Authentication Providers

Control Flag Value	Description
Required	The authentication provider is always called, and the user must always pass its authentication test. Regardless of whether authentication succeeds or fails, authentication proceeds down the list of providers.
Requisite	The user is required to pass the authentication test of the authentication provider. If the user passes the authentication test of the authentication provider, subsequent providers are executed but can fail (except for authentication providers with the JAAS control flag set to required).
Sufficient	The user is not required to pass the authentication test of the authentication provider. If authentication succeeds, no subsequent authentication providers are executed. If authentication fails, authentication proceeds down the list of providers.
Optional	The user is allowed to pass or fail the authentication test of the authentication provider. However, if all authentication providers configured in a security realm have the JAAS control flag set to optional, the user must pass the authentication test of one of the configured providers.

Note: When an authentication provider is added to an existing security configuration, by default the control flag is set to required.

Credentials in Single Sign-on

Created as a direct result of authentication, credentials are usually time-limited objects that you can use to perform tasks such as authorization checks, attribution requests, and single sign-on (SSO).

In addition to principals, credentials help identify users to other remote services. For example, in Sybase Unwired Platform where CSI is integrated as the primary security system, credentials perform single sign-on into other systems that are accessed to fulfill a client's request. CSI provides a wrapper on the credential concept to associate a name with a

credential. This is to aid Unwired Server in selecting the correct credential to propagate to the remote service when multiple providers (or multiple instances of a provider) add credentials of the same type (class) to the JAAS subject.

You can configure the endpoint connection definition with a property to identify the NamedCredential that should be propagated to the remote service to perform SSO. Credentials can also be used by the providers to communicate session information. An authentication provider can store the session information necessary for the attribution and authorization providers to perform their functions.

Add credentials using the `getPublicCredentials()` and `getPrivateCredentials()` methods from the Subject class. CSI does not treat public and private credentials differently. Providers should follow standard JAAS best practices when using credentials. When the security context is destroyed using the `destroy()` method, each of the public and private credentials that implements the `javax.security.auth.Destroyable` interface has its `destroy()` method called. This can be used to close any security doors left open by the credential's existence. For example, destroying the credentials may terminate a user's session that was kept open and stored in the credential for use by the authorization or attribution provider, or it may have the eventual effect of triggering a logout audit record in the underlying security system.

The interface `com.sybase.security.SSOTokenCredential` should be implemented by credentials that contain tokens to be used for single sign-on into an SAP® system. It is used to identify the credential to be forwarded to an SAP endpoint (both JCo and DOEC endpoints).

The interface `com.sybase.security.CertificateCredential` should be implemented by credentials that contain the certificate to be used for single sign on into an SAP system. It is used to identify the credential to be forwarded to an SAP endpoint (both JCo and DOEC endpoints).

The interface `com.sybase.security.NamedCredential` associates a name with a credential object so the Web service endpoints can be configured with the name of the credential to forward to the Web service for single sign-on.

The interface `com.sybase.security.core.ExpiringCredential` allows a credential to be marked with an expiration time. This is used by Sybase Unwired Platform server to expire the user session in the authentication cache based on the credential expiration time instead of the configured authentication cache timeout interval.

Authorization

The authorization provider interface is defined by the `com.sybase.security.provider.Authorizer` interface.

There are two primary worker methods:

- `checkRole()`

- `checkAccess()`

There are also two security context life-cycle methods, `initContext()` and `destroyContext()`, and one provider lifecycle method, `init()`, all of which are inherited from `com.sybase.security.provider.SecContextProvider`.

Provider Initialization

Authorization providers are normally initialized once per static class context. The provider instance itself is shared among any security context instances subsequently created.

The provider must expect to be used concurrently by multiple security clients. The security provider is initialized by the CSI infrastructure with a call to the `init()` method, which takes as an argument a `java.util.Map` of configuration data specific to the provider.

The provider should take this opportunity to validate connections to external resources, if possible, so that configuration errors are manifested as early as possible in the business flow. After initialization, the provider list is provided to security context instances as they are being created. During creation, a security context calls each provider's `initContext()` method, passing in an internal context `java.util.Map` (not to be confused with the security context). This structure should be used to store any working information that the provider needs to maintain state. Because the provider instances are used concurrently by multiple clients, each provider method's first argument is this context object that is unique to the client's security context.

After context initialization has completed, the following context map data is passed into the security provider methods:

Table 2. Context Map Data

Constant	Expected Data Type	Description
`ProviderConst.SEC_CONTEXT`	`SecContext`	The client security context associated with the context map.
`ProviderConst.CURRENT_SUBJECT`	`SecSubject`	If authenticated, the subject associated with this security context. A provider can use this object to retrieve principals and credentials from the underlying JAAS subject.
`ProviderConst.WARNING_MANAGER`	`WarningManager`	Providers may use this object to add warnings for the current operation. **Note:** Deprecated, use `ProviderServices`.

Constant	Expected Data Type	Description
`ProviderConst.PROVIDER_SERVICES`	`ProviderServices`	Any service exposed internally to all providers is available through this interface, for example `WarningManager`, access to profiles, certificate validation, and so on.

Authorization Checks

The authorization capabilities in Sybase Unwired Platform center around two primary authorization methods: role checks and resource access checks.

Role checks are performed using the context's `checkRole()` method, passing the role ID as the parameter. A list of potential roles that may be used for access checks can be retrieved using the `listRoles()` method. Some provider sets have the ability to enumerate the roles available. The return value of this method does not necessarily comprise the complete list of roles, depending on the providers.

Resource access checks are more complicated than role checks because three factors are applied to resource access checks:

- the resource, represented by the `SecResource` interface
- the action, which is a named object
- the subject

Before attempting an access check, a reference to a resource object must be retrieved through a call to `getResource()`, possibly preceded by a call to `listResources()` to obtain a resource list. Next, the client can either immediately call `checkAccess()` supplying the resource reference and an action identifier, or the client can first use the `listActions()` method to retrieve a list of potential actions that can be expected to be performed on a given resource instance. As with all enumeration methods, providers are not required or guaranteed to implement these and in this case additional valid actions may exist for the given resource.

The calls to the variety of `SecContext.checkAccess()` and `SecContext.checkRole()` methods by the client result in calls to the corresponding methods in the authorization provider. The security context combines the results of the role and access check from all of the configured authorization providers when delegating these calls. Therefore, because one provider identifies a user as having a particular role, it does not mean that the `checkRole()` call succeeds.

Access Check

The access checks performed by the `checkAccess()` method are similar to role checks.

There are two differences regarding access checks:

- Instead of a role name, a `SecResource` instance and an action string are supplied. The `SecResource` object is retrieved with the help of the attribution provider, so it is not possible to perform access checks without an attribution provider installed.
- A `com.sybase.security.SecEnvironment` instance may be present, depending on which variant of the method was called.

This argument is additional information the client must pass to the back-end security provider. This could be information about where the security check occurs or extra rules to evaluate in addition to the standard security check. The contents of this structure are not defined by the CSI except to declare it as attributed. Clients and providers need to coordinate what attributes are supported and meaningful. It is important for authorization providers to know when to return NO and when to return ABSTAIN. The same decision flow should be used to determine the return value for the `checkAccess()` method.

Provider Implementation of Role Check

The `checkRole()` method accepts three arguments: the context map, the role name, and a `SecSubject` object.

The first argument is the context map. This is the same argument that was passed into the `initContext()` method, so any information saved in that method call is now available using this object instance.

The second argument provided to the `checkRole()` method is the role name. The specifics of how the implementation interprets this string argument are determined during implementation. However, Sybase strongly recommends that authorization and attribution providers coordinate their efforts. This ensures the results from an attribution provider's `listRoles()` method return named objects whose identifiers are supported as role name arguments in the authorization provider's `checkRole()` method. It is possible for the provider's `checkRole()` implementation to accept multiple role-name formats beyond those returned by the attribution provider.

The third argument provided to the `checkRole()` method is a `SecSubject` object. Note that this is not necessarily the subject object that is authenticated into the active security context, which can be retrieved from the first argument (context map). In CSI, it is possible for a security client to perform a role check on another security subject. Not all providers need support this capability – if the provider does not, then it should simply return ProviderConst.ABSTAIN.

There are three valid return values from an authorization provider authorization check (both role and access check), all of which are contained in the `com.sybase.security.ProviderConst` interface. YES, NO, and ABSTAIN are the three constants. In general, if a provider can verify the existence of a role and can verify that the specified `SecSubject` does not have the role, the provider should return NO. Otherwise, the provider should return ABSTAIN.

For example, the `RoleCheckAuthorizer` provider provides role checking capabilities when the login module adds principals and credentials that implement the `RoleCheck`

interface. This provider checks if a principal or credential exists in the JAAS subject with the specified role name and that it implements the `RoleCheck` interface. If the provider finds one, it returns YES. Note that the absence of a principal or credential does not guarantee that the role is not granted to the authenticated user, because the user might be granted the role through an alternate provider in a mixed security model. In this case, the provider must return ABSTAIN or lose interoperability with other authorization providers.

Attribution

The attribution capabilities provide attribute information for attributed objects within the security system and provide lists of objects.

The attribution capabilities are classified as those that perform one or both of the following tasks:

- Provide complete attribute information for attributed objects, for example resources and subjects, within the security system
- Provide enumerated lists of objects, for example resources, resource types, actions, and roles, possibly filtering using qualifiers

Attributed objects are represented by the `com.sybase.provider.Attributed` interface, which is sub-classed by several core interfaces such as `SecSubject` and `SecResource`.

In general, attributed objects may have zero or more attributes assigned to them represented by string keys. Each of these keys may have zero or more string values associated with it. Example attributes for a subject may be first name and e-mail address. Each attribution provider defines their own list of attributes, keys, and values. Some standard attributes that all attribution providers should support, if relevant, are:

- `Const.ATT_NAME`
- `Const.ATT_DESCRIPTION`

These map cleanly to extra fields in named objects.

Not all provider sets include an attribution provider. For example HttpAuthetnicationLoginModule and NTProxyAuthLoginModule do not have a corresponding attributer. In Sybase Unwired Platform, the attributer is the key interface for making changes to the back-end security system. The attributer is typically used for listing resources and roles in the back-end. For example LDAPAttributer is used to list the roles defined in the LDAP server. Using the attributer, you can implement self-registration to create a user account in the back-end security configuration or to change a password.

The attribution provider interface is defined by the `com.sybase.security.provider.Attributer` interface. The attribution providers have the same life-cycle methods as the authorization providers. The same rules should be followed with respect to thread safety. All of the attribution provider methods include a context map which can be used to store state information associated with a security context.

Enumeration Methods

Enumeration of objects is a function of the attribution layer.

Methods that meet this qualification typically return a list of named objects and have a name in the format listXYZ where XYZ is an object type, such as Roles or Actions. The enumeration APIs are those which are prefixed by list and return a java.util.List.

Order is important and relevant within the enumeration methods. CSI calls the providers in the order in which they are configured. Values in lists are concatenated in the order in which providers are called. In some cases, duplicates are removed.

Attribution Methods

There are three attribution methods that an attribution provider should implement: `attributeAuthenticatedSubject()`, `attributeSubject()`, and `attributeResource()`.

After authentication, the method `attributeAuthenticatedSubject()` is called to further elaborate on a subject's attributes. Because authentication is done by combining JAAS LoginModules and the results of this authentication process are simply a set of principals and credentials, there is more required after authentication to provide additional information about a user. Immediately following authentication, the CSI architecture reviews the authenticated subject's principals and initializes the ID and NAME attributes of the `SecSubject` object as described in *Authentication*.

Any attributes other than these must be provided by the `attributeAuthenticatedSubject()` method, which takes two arguments. The first argument is the context map. The second argument is the `SecSubject` object that needs to have attributes defined on it. The attribution provider should use information contained in this object to add more attributes to the subject object.

For example, an LDAP authentication module adds an `LDAPDNPrincipal` principal to the subject's principal set. A companion LDAP attribution provider may contain the `attributeAuthenticatedSubject()` implemented to look for the custom `LDAPDNPrincipal` object in the principal set. It then uses the DN stored in the principal to connect to the LDAP server and retrieve extended attributes about the user represented by the principal. The specifics of the attributes are undefined in the context of CSI – each attribution provider defines its own sets of attributes. The only subject attributes which are used in Sybase Unwired Platform are ID and NAME attributes. Others are ignored.

While the most common way to retrieve a `SecSubject` instance is through an authenticated security context's `getSubject()` method, there is also a way to retrieve an unauthenticated subject using the context's `getSubject(String id)` method. It is through this method that the attribution provider's `attributeSubject()` method is called. This method is similar to the `attributeAuthenticatedSubject()` method, except the method is supplied with a single identifier in a third argument that is to be used as the subject identifier. If

an attribution provider can recognize and attribute a subject with the given identifier, it should return "true" when complete. This indicates to CSI that the attribution provider recognized a user with the specified identity, and that it should pass an unauthenticated `SecSubject` instance back to the CSI client.

The third attribution method, `attributeResource()` is used in a manner similar to `attributeSubject()`. The security context's `getResource()` method delegates to each of the configured attribution providers, and if one or more signals that it recognized and attributed the provider, a `SecResource` implementation is returned to the client. One difference is that in addition to attributes, this method should populate the resource type lists for the `SecResource` object.

Multiple Provider Interaction

When multiple active providers are configured, CSI reconciles conflicting decisions across the providers.

Often a configuration may include a variety of active providers, all interacting and voicing their own perspective on an incoming request. CSI reconciles the conflicting decisions.

Authentication is unique in that it is partly the responsibility of the individual providers and partly the responsibility of the CSI configuration. CSI uses the JAAS authentication model for authentication and allows each provider to be associated with a control flag (required, requisite, sufficient, and optional). See *http://www.oracle.com/technetwork/java/javase/jaas/index.html* for a introduction to the JAAS model. When reviewing the JAAS model, consider references to `LoginContext` the same as `SecContext` because CSI provides its own entry point.

An additional feature the framework adds on top of the JAAS model is the ability to detect impersonation. This is meaningful when both a user name and other data (certificate or an opaque token) are provided in an authentication request. When the authentication succeeds, the authenticated subject's attributes contain the `Const.ATT_NAME` attribute with the value set to the set of names (derived from the Principal objects of type `com.sybase.security.provider.SecNamePrincipal`) aggregated from the configured providers.

If the configured providers only consider the extra data (for example, certificate or token) and add Principal objects based on that, the supplied user name may not match any of the values for the `Const.ATT_NAME` attribute of the authenticated subject. When it is critical for the supplied user name to match at least one of the Principal names, you can set the configuration property checkImpersonation to true. This results in authentication failure even when all the configured login modules succeed but none of the Named principal names match the supplied user name.

in CSI, the result of authorization operations is either YES, NO, or ABSTAIN. However, there may be multiple authorization providers all vying to respond to a given authorization request. Providers have three possible answers to the authorization question – true, false, or abstain.

Abstain should be used when a provider is not capable of answering yes or no. CSI denies the authorization request if either of the following occurs:

- Any authorization provider returns NO
- No authorization provider returns YES

CSI permits the authorization request in any other situation. Another way of describing a successful authorization request is when at least one authorization provider returns YES and none return NO.

The result of attribution operations is also a combined exercise. For methods such as enumeration methods, the union of the corresponding values for each provider is created and returned to the client. For attributed objects, the union of the attributes that each provider states are created.

For example, Provider A knows that Resource A has a NAME attribute of "Resource A" and an OWNER attribute of "Bob". Provider B knows that resource A has a NAME attribute of "Resource A-B" and a GROUP attribute of "Sales". The net attribute names and values for Resource A would be NAME = {"Resource A", "Resource A-B"}, OWNER="Bob", GROUP="Sales".

Shared Data Structures for Providers

Provider instances use a shared map to store information and share it with other providers.

All the provider instances are passed a shared map when context-specific methods are invoked. This map can be used by the provider instances to store context-specific information and share it with other providers that need to access it. If providers need to store and share data at the factory level so they can access the data across multiple contexts, they can do so by storing the data in the map indexed with the key `ProviderConst.FACTORY_LEVEL_SHARED_STATE`.

When using the core provider CertificateValidationLoginModule or CertificateAuthenticationLoginModule, the validated certificate chain is stored in the shared map and available to all the authentication providers using a pre-defined key. Providers can set and retrieve the certificate chain using the key `ProviderConst.CERTIFICATE_SHARED_KEY` to support certificate authentication principals.

In Sybase Unwired Platform, all the client parameters (including personalization parameters as well as the HTTP headers and cookies from the client session) are populated in the shared map passed into the `initialize()` method. A map is stored in the shared state with the key `ProviderConst.HTTP_PROPERTIESCOOKIES_SHARED_KEY`. It contains the client parameter names as the keys.

Capability Query

CSI provides the capabilities API to allow a CSI client to interrogate the capabilities of the underlying providers within CSI.

A capability query result has a Boolean value, either true or false. Capabilities are dynamic and may change over time even within the same security context. Standard capabilities are defined in the `com.sybase.security.Const` class. Some of these are listed in the table below:

Table 3. CSI Capabilities APIs

Capability Name	Description
`com.sybase.security.capabilities.provider.SelfRegistration`	If the security context has the ability to perform self registration, this capability is available. **Note:** Not used in Sybase Unwired Platform.
`com.sybase.security.capabilities.provider.X509Authentication`	If the security context is able to authenticate X.509 certificates, this capability is available.
`com.sybase.security.capabilities.provider.PasswordChange`	Not used in Sybase Unwired Platform.
`com.sybase.security.capabilities.provider.FineGrainAccessControl`	The provider implements the `checkAccess` methods, not just the `checkRole` methods.

The capability API supports provider-specific capabilities. This allows a provider to define its own set of capabilities that may be checked by Sybase Unwired Platform. Providers can implement the `com.sybase.security.provider.SecProviderCapabilites` interface to respond to a capabilities query.

Capabilities API that may be implemented by providers is as follows:

```
package com.sybase.security.provider;

/**
* Optional provider interface that allows providers to respond to
capability requests.
*/
public interface SecProviderCapabilities
{
/**
* Called when when building the capability set of a provider. The
* result of this call will not be cached by the SecContext
implementation.
* @param context the context map
* @param capability the capability to check
* @throws SecException if some sort of error occurs that should abort
```

```
* the entire capability query.
*/
boolean hasCapability(Map<String, Object> context, String
capability) throws SecException;
}
```

For example, a method implementation in a provider implements `SecProviderCapabilities` interface and supports certificate authentication capability:

```
public boolean hasCapability(Map<String, Object> context, String
capability) throws SecException
{
if (capability.equals(CAPABILITY_X509_AUTHENTICATION))
{
return true;
}

return false;
}
```

CSI Audit Generation and Configuration

CSI provides audit trail generation and configuration.

An audit destination may write an audit record to a file. All audit destinations have the ability to use an audit formatter, although they may be configured to ignore this if they do their own formatting. Audit destinations may be configured to initialize a default formatter. Audit destinations must be thread safe, unlike other CSI providers. CSI guarantees that each audit destination is created only once per factory instance.

However, it may be desirable for an audit destination to be aware of other instances of itself instantiated in the same VM and ensure that these instances appropriately share access to common resources, such as audit files. Audit destinations may also add additional attributes to the audit record, such as a sequential record ID, to guard against log tampering or signature attributes that can be later verified to check for audit record tampering.

The primary sources for generating audit records are:

- CSI core classes perform a variety of operations that generate audit records. Examples of events that are audited include authentication and authorization decisions, self registration attempts, and configuration information such as active provider sets.
- Providers may audit additional information about the operations to which they contribute. An example of extra information might be an extended response from the authentication server that indicates why an operation failed in more detail.

Providers that need to issue audit records can take advantage of auditing APIs defined in the `com.sybase.security.providers.ProviderServices` interface. Those audit APIs require the provider pass in an audit token. This token identifies which provider is issuing the audit request. The provider can retrieve and save this token from the provider configuration using the constant

`ProviderConst.AUDIT_TOKEN_CONFIG_PROPERTY` in the provider's `init()` method or a LoginModule's initialize method.

An audit record includes five common parts:

- timestamp - the time at which the event occurred
- resource class - the scope used when processing the resource ID
- resource ID - the object on which an operation is being performed
- action - the operation that is being performed
- decision - the result of the security check

The decision has four potential values: PERMIT, DENY, ABSTAIN or NOTAPPLICABLE. The NOTAPPLICABLE decision is used for those cases where the audit is an event rather than a security decision.

In addition to the core of the audit record, there are additional attributes. These attributes vary with the type of audit record being generated but may include context ID, provider identifier, and failure reason.

The auditing providers consist of the following configurable components:

- The `com.sybase.security.provider.AuditDestination` interface defines APIs for the audit destination component, which decides where to write audit record. The framework includes a simple file-based audit destination component, `com.sybase.security.core.FileAuditDestination`, that uses an audit formatter to retrieve a string representation of the audit record, and writes it to a file.
- The `com.sybase.security.provider.AuditFilter` interface defines APIs for audit filtering components, which decide whether an audit record should be audited or not. The default filter component, `com.sybase.security.core.DefaultAuditFilter`, adopts a filter string to perform audit record filtering. The syntax for the filter consists of zero or more filter expressions, delimited by parenthesis (brackets denote optional values). For example, (expr1)[(expr2)...]. Each of these expressions has syntax like the following: [key1=value [,key2=value...]]. The allowed keys are: ResourceClass, Action, and Decision. For example, filter string (ResourceClass=core.*,Decision=Deny) enables auditing all of the CSI core resource classes involved in a deny decision.

The default formatter, `com.sybase.security.core.XmlAuditFormatter`, stores the audit record in one type of XML format. Each audit destination component is associated with one audit filter and one audit formatter. The framework supports an audit destination control flag scheme similar to Java's JAAS LoginModule control flags of required, requisite, sufficient, and optional.

Table 4. Control Flag Settings for AuditDestination

Control Flag Value	Description
Required	If the audit filter associated with the destination enables logging a particular event, the audit record is always logged to the audit destination, and the logging must succeed. Regardless of whether or not the audit record is successfully logged to the destination, the other configured audit providers are invoked.
Requisite	If the audit filter associated with the destination enables logging a particular event, the audit record is required to be successfully logged to the audit destination. If the audit record is successfully logged to the audit destination provider, subsequent providers are executed but can fail (except for providers with the JAAS control flag set to required).
Sufficient	If the audit filter associated with the destination enables logging a particular event, the audit record is not required to be successfully logged to the audit destination. However, if the audit record is successfully logged by the provider, no subsequent providers are executed. If audit record logging fails, the audit filter proceeds down the list of configured providers.
Optional	If the audit filter associated with the destination enables logging a particular event, the logging of the audit record by the provider is allowed to succeed or fail. However, if all the audit destination providers are configured with control flag set to optional, the audit record must be successfully logged by at least one provider.

Note: The audit results are combined in the same way authentication results of login modules are combined to decide if the authentication succeeds or not overall. If the combined audit result fails, then the security operation being audited fails. For example, if an authorization check is audited and the audit filter is set up to log that event, the authorization check fails if the audit record cannot be successfully logged even if the authorization provider returns VOTE_YES.

Configuration Validation

The provider architecture allows each configured provider to have their own namespace of configuration options.

When a provider's `init()` method (or a login module's `initialize()` method) is called, a configuration map is provided. The keys and values in the map always appear as string types.

To detect any configuration errors before they are saved, CSI validates that the supplied internal configuration in the properties format adheres to the stored provider metadata, and the providers validate the supplied configuration by performing runtime checks where possible. The provider implementations should include a `sybcsi-provider.xml` file that

contains the metadata about the valid configuration options (including names, types, default values, and whether they are required or optional properties, and so on) to simplify the configuration administration and validation. The provider metadata should conform to the latest configuration XSD included in the SDK.

A provider can participate in configuration validation by implementing the optional interface `com.sybase.security.provider.SecConfigurationValidatingProvider` to perform the same validation checks on the specified configuration options that it performs at runtime when instantiated with the configuration using the `init()` method. It should return the validation errors as a list of `com.sybase.security.provider.ConfigurationProblem`.

A configuration problem report includes an error description, its severity, and the configuration property with which it is associated. In the case of a missing required property or an invalid property combination, the problem is associated with the provider itself.

Login modules can implement `com.sybase.security.provider.NamedCredentialProvider` as an optional interface to aid in validating that the NamedCredential added by the provider does not conflict with the credentials added by other configured login modules.

Internationalization in CSI

Internationalization ensures that data integrity is maintained while passing through the CSI core.

The core CSI classes do not provide any character set conversions. All string type values are expected to be encoded in Java standard UCS-16. Any providers that receive or send character-based information in any other character set are responsible for character set conversions.

The following examples show how the API handles character sets:

XML Configuration

The XML configuration provider retrieves configuration data from the XML file. It uses standard Java APIs to read and process the XML data. Any non-ASCII data embedded in the XML file is properly handled and converted to strings using an active Java XML parser (JAXP).

LDAP Attributer

The LDAP attributer uses the JNDI LDAP provider to retrieve attribute data about specific subjects. Exact string references are returned from the JNDI LDAP provider, and the CSI core does not inspect the string.

Login Module

The login module requests any credentials from the supplied callback. The transmission of credentials to the login module is done through the JAAS callback mechanism, requiring the login module to request any credentials from the supplied callback.

CSI and Provider Localization

Message localization within the CSI core and the default providers is performed using Java resource bundles along with some internal-only helper classes.

Messages are localized to the system locale before being written to a file or shown to a user.

Note: No translations are maintained, except the English default.

CSI core does not provide localization services to the providers. Provider authors are encouraged to use the Java resource bundle technique to provide localization services, but may choose to use other methods.

CSI Core Java Logging

CSI core and its default providers use Java logging APIs.

Sybase Unwired Platform using the CSI core enables you to configure and localize logging.

CSI does not provide any logging services to providers. It is the responsibility of each provider author to log messages using their framework of choice. Providers are encouraged to use Java logging and resource bundles, which ensure that custom provider logging can be configured from Sybase Control Center for Sybase Unwired Platform.

Sybase Unwired Platform Logging Configuration

The Sybase Unwired Platform logging configuration sets the logging level for the core CSI framework classes and the default providers shipped with Sybase Unwired Platform when the **Security** component logging level is set in Sybase Control Center for Sybase Unwired Platform.

If you develop and deploy a custom provider with a different namespace, the log level for it can be set using the **Other** component in the Sybase Control Center for Sybase Unwired Platform log configuration screen. The **Other** component includes all third-party libraries deployed and used in Sybase Unwired Platform, and results in a large number of log messages.

Sybase recommends that a custom security provider be developed in the namespace `sup.custom.security.*`.

Logging Localization

All log messages generated at the INFO level and higher within the framework and the default providers can be localized.

Localization of messages within CSI core and the default providers is performed using Java resource bundles along with some internal-only helper classes. In contrast, DEBUG messages cannot be localized because they are intended to be interpreted by technical support and development personnel only. Implement a custom provider implementation using the same guideline.

For more information, see *Localization*.

Error Handling for Providers

Errors in the providers are typically reported by throwing exceptions.

Localized error messages are made available from the `java.lang.Throwable.getMessage()` method that all exception classes implement. However, the framework defines a few exception types and interfaces that are intended to be used for reporting errors. The exception `com.sybase.security.SecException` can be thrown from most methods.

The framework aggregates results from multiple providers where each can throw their own errors for a given operation. Also, the providers can be stacked where only a subset of providers are required to fulfill a request (authentication or audit) controlled by the control flag. For example, if the first authentication provider fails with an exception, and the second one succeeds, the exception thrown by the first provider is not propagated to the client if the control flag dictates that the error is irrelevant in the authentication process. Control flag is a JAAS concept. For more detailed information, refer to the `javax.security.auth.login.Configuration` javadoc at *http://docs.oracle.com/javase/6/docs/api/javax/security/auth/login/Configuration.html*.

The framework allows all warnings to be tracked and retrieved so they can be propagated to Sybase Unwired Platform clients. Warnings are represented by the `com.sybase.security.SecWarning` interface. Providers can introduce new warnings using special provider-side APIs. There are several predefined warning sub-interfaces for standard security messages such as `password is expiring in the future at this time/date`.

See the javadoc for more details. Sybase Unwired Platform inspects the warnings after an authentication attempt and although not all warnings are returned to the client, Sybase Unwired Platform does look for a few pre-defined warnings that are propagated to clients.

Error Localization

Exception messages generated by the security framework and the default providers can be localized.

All messages are propagated in the system locale only. The security context does not provide a way to specify an alternative location for message translation

CSI does not provide error or message localization services to the providers. Each provider author must choose a localization facility. Providers are encouraged to use Java resource bundles for error or message localization.

Reporting Errors and Warnings from Providers

Providers indicate errors and warnings in a variety of ways, including framework and stack trace logging to the logging system. Provider authors can customize exception handling and reporting.

Nearly all provider methods include `SecException` to indicate a failure. At a minimum, the framework logs the exception's message and stack trace to the logging system. The framework may propagate the message to the client, depending on the situation. Propagation makes exceptions accessible to the clients. The framework can add or log exceptions to the context warning list for troubleshooting purposes.

On the provider side, these interfaces enable adding warnings to the context that can be retrieved by the client:

```
package com.sybase.security.provider;

public interface WarningManager {
    void addWarning(SecProvider provider, SecWarning warning);
    void addWarning(javax.security.auth.spi.LoginModule provider,
                SecWarning warning);
}

public interface ProviderConst {
    ...
   public static final String WARNING_MANAGER = " CSI.warningManager
";
    public static final String PROVIDER_SERVICES =
"CSI.providerServices";
}
```

A provider can add a warning to the context by retrieving the warning manager or provider services from the context map. This example shows how a login module can add a warning of `you've been warned` from the login method:

```
public class ExampleLoginModule implements
javax.security.auth.spi.LoginModule {
    private Map context;

    public void initialize(Subject subject,
```

```
                CallbackHandler callbackHandler,
                Map sharedState,
                Map options) {
        context = sharedState;
    }

    public boolean login()throws LoginException
         {

        ProviderServices providerSvcs =
            (ProviderServices)context.get(ProviderConst.
PROVIDER_SERVICES);

        providerSvcs.addWarning(this,
         new SimpleWarning("you've been warned"));


        return true;
    }

    // ... other methods snipped for brevity
}
```

Use the same techniques from all other provider types, as well as the supplied context map. When a `LoginException` is thrown from the login method of an authentication provider, the framework automatically adds a warning to the warning list. If the `LoginException` instance already implements the `SecWarning` interface, the exception itself is added as a warning. Otherwise, the exception is wrapped in a lightweight wrapper (`com.sybase.security.provider.SecLoginExceptionWarningImpl`).

A provider can use the `com.sybase.security.provider.SecLoginExceptionAuthenticationFailureWarningImplclass` to simultaneously signify login failure and supply a more specific failure reason mapped from `com.sybase.security.core.AuthenticationFailureReasons`.

For more details, see the javadoc for these classes.

Security API Reference

Use the Quick Start Guide and the javadoc as a Security API reference.

Refer to the sample project in the *Quick Start Guide* located in the Sybase Unwired Platform installation directory: *SUP_HOME*`\Servers\UnwiredServer\ServerSDK\securityAPI`

Refer to the Security API javadoc for applicable documentation for each available package.

- View this in HTML format: *Security API Reference*.

- View this in PDF format: *Security API Reference*.

Management API

This section provides information about using the Sybase Unwired Platform Management API 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 section describes the features and usage of the Management 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.

Introducing Management API

Use the Management API in Sybase® Unwired Platform 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.

Management 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 Management 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 Management API.

Companion Documentation for Management API

Companion guides include:

- *System Administration*
- *Sybase Unwired WorkSpace – Mobile Business Object*
- *Troubleshooting*

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

Management API Javadoc

The Management API installation includes javadoc. Use the Unwired Server javadoc 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 these APIs. By default, javadoc is installed to *SUP_HOME*`\Servers\UnwiredServer\AdminClientAPI \com.sybase.sup.adminapi\docs\api\index.html`.

The top left navigation pane lists all packages installed with Sybase 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

Sybase® Unwired Platform documents are available for administrative and mobile development user roles. Some administrative documents are also used in the development and test environment; some documents are used by all users.

See *Documentation Roadmap* in *Fundamentals* for document descriptions by user role.

Check the Sybase® Product Documentation Web site regularly for updates: *http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories* , then navigate to the most current version.

Management API Changes in Version 2.2

Changes in the Management API between 2.1 and ESD #3 and 2.2 include `SUPCluster`, `SUPConfiguration`, `SUPServerConfiguration`, `SUPRelayServer`, `SUPDomain`, `SUPMobileHybridApp`, `SUPApplication`, and `SUPSecurityConfiguration`

For Management API changes prior to 2.1 ESD #3, see *Release Bulletin 2.1 ESD #3* on Product Documentation, the *Administration Client API Changes* topic: *http://dcx.sybase.com/index.html#sup0213/en/com.sybase.infocenter.dc00835.0213/doc/html/aro1345156276263.html*

SUPCluster API Changes

API changes for the `SUPCluster` interface, used to manage the cluster to which the Unwired Server instance belongs.

Table 5. New SUPCluster Methods

Methods	Description
• `getHttpProxySetting` • `updateHttpProxySetting`	Manage HTTP proxy settings
• `getServerLogSetting()` • `updateServerLogSetting()`	Manage the Unwired Server logs
• `getBESResponsePortioningLimit` • `setBESResponsePortioningLimit`	Manage the response portioning limit
• `isHttpLogEnable()` • `setHttpLogEnable` • `getMaxHttpLogFileSize()` • `setMaxHttpLogFileSize` • `getHttpLogFileName()` • `setHttpLogFileName` • `isSeparateHttpLogFile()` • `setSeparateHttpLogFile` • `isReuseHttpLogFile()` • `setReuseHttpLogFile` • `getHttpLogArchiveFileName()` • `setHttpLogArchiveFileName` • `isArchiveHttpLog()` • `setArchiveHttpLog` • `isCompressHttpLogArchive()` • `setCompressHttpLogArchive`	Manage the HTTP log settings

Table 6. Deprecated SUPCluster Properties

Property	Description
`sup.admin.maxThreads`	The maximum number of concurrent threads for the management port.
`sup.admin.iiops.maxThreads`	The maximum number of concurrent threads for the secure management port.

Documented in: *Developer Guide: Unwired Server Runtime*, see *Managing Clusters*

SUPConfiguration API Changes

API changes for the `SUPConfiguration` interface, used to manage the Sybase Unwired Platform configuration.

Table 7. New SUPConfiguration Methods

Methods	Description
• `getContext` • `commitClusterConfiguration` • `refreshClusterConfiguration`	Manage the cluster configuration
• `getManagementConfiguration` • `updateManagementConfiguration`	Manage management configurations
• `enableOCSPConfiguration` • `getOCSPConfiguration` • `updateOCSPConfiguration`	Manage the Online Certificate Status Protocol (OCSP) configuration
• `getPerformanceConfiguration` • `updatePerformanceConfiguration`	Manage the performance configuration
• `getConfigurationCache` • `updateConfigurationCache` • `commitConfigurationCache` • `refreshConfigurationCache`	Manage the cache configuration
• `getReplicationConfiguration()` • `updateReplicationConfiguration()`	Manage replication configuration

Methods	Description
• `getMessagingConfiguration()` • `updateMessagingConfiguration()`	Manage messaging configuration
• `getSolutionManagerConfiguration()` • `updateSolutionManagerConfiguration()`	Manage the Solution Manager configuration
• `getDCNConfiguration()` • `updateDCNConfiguration()`	Manage data change notification (DCN) configuration
• `getWebContainerCommonConfiguration` • `updateWebContainerCommonConfiguration` • `commitWebContainerConfiguration` • `refreshWebContainerConfiguration`	Manage the Hybrid Web Container configuration
• `getHTTPListenerConfigurations` • `addHTTPListenerConfiguration` • `deleteHTTPListenerConfiguration` • `updateHTTPListenerConfiguration`	Manage HTTP listener configurations
• `getSSLSecurityProfileConfigurations()` • `addSSLSecurityProfileConfiguration()` • `deleteSSLSecurityProfileConfiguration()` • `updateSSLSecurityProfileConfiguration()`	Manage the SSL security profile configuration
• `getKeyStoreConfiguration` • `updateKeyStoreConfiguration`	Manage the keystore configuration
• `getTrustStoreConfiguration` • `updateTrustStoreConfiguration`	Manage the truststore configuration
• `refresh` • `commit`	Manage the server configurations locally

Documented in: *Developer Guide: Unwired Server Runtime*, see *Managing Clusters*

SUPServerConfiguration API Changes

API changes for `SUPServerConfiguration`, used to administer the Unwired Server. With this release, most `SUPServerConfiguration` methods have been deprecated, except the `OutboundEnabler` and JVM-related API.

Table 8. New SUP Server Configuration Methods

Methods	Description
• `startOutboundEnablers` • `stopOutboundEnablers` • `getOutboundEnabler` • `getOutboundEnablers` • `saveOutboundEnabler` • `updateOutboundEnabler` • `deleteOutboundEnabler` • `deleteOutboundEnablers` • `addOutboundEnablerCertificates` • `getOutboundEnablerCertificates` • `deleteOutboundEnablerCertificateFile` • `deleteOutboundEnablerCertificateFiles`	Manage outbound enablers.

Table 9. Deprecated SUP Server Configuration Administration Methods

Methods	Description
• `getReplicationSyncServerConfiguration()` • `updateReplicationSyncServerConfiguration()`	Manage the replication synchronization server configuration
• `getMessagingSyncServerConfiguration()` • `updateMessagingSyncServerConfiguration()`	Manage the messaging synchronization server configuration
• `getConsolidatedDatabaseConfiguration()`	Manage the viewing of the consolidated database configuration

Methods	Description
• `getAdministrationListenerConfiguration()` • `updateAdministrationListenerConfiguration()`	Manage the administration listener configuration
• `getHTTPListenerConfigurations()` • `addHTTPListenerConfiguration()` • `deleteHTTPListenerConfiguration()` • `updateHTTPListenerConfiguration()`	Manage the HTTP listener configuration
• `getSecureHTTPListenerConfigurations()` • `addSecureHTTPListenerConfiguration()` • `deleteSecureHTTPListenerConfiguration()` • `updateSecureHTTPListenerConfiguration()`	Manage the HTTPS listener configuration
• `getSSLSecurityProfileConfigurations()` • `addSSLSecurityProfileConfiguration()` • `deleteSSLSecurityProfileConfiguration()` • `updateSSLSecurityProfileConfiguration()`	Manage the SSL security profile configuration
• `getKeyStoreConfiguration()` • `updateKeyStoreConfiguration()`	Manage the key store configuration
• `getTrustStoreConfiguration()` • `updateTrustStoreConfiguration()`	Manage the truststore configuration

Methods	Description
• getApplePushNotificationConfigurations() • addApplePushNotificationConfiguration() • deleteApplePushNotificationConfiguration() • updateApplePushNotificationConfiguration()	Manage Apple native notification configuration

Table 10. Deprecated SUPServerConfiguration Methods

Methods	Description
• ReplicationSyncServer • ReplicationNotifier_Push • ReplicationPushNotificationGateway	Manage the replication synchronization server
• MessagingSyncServer	Manage the messaging synchronization server
• AdministrationListener • SecureAdministrationListener • HTTPListener • SecureHTTPListener	Manage the administration listener
• SSLSecurityProfile • TrustStore • ReplicationNotifier_Pull • OCSP	Manage metadata-based server components
• getSSLSecurityProfileConfiguration • addSSLSecurityProfileConfiguration • deleteSSLSecurityProfileConfiguration • updateSSLSecurityProfileConfiguration	Manage the security profile configurations
• getKeyStoreConfiguration • updateKeyStoreConfiguration	Manage the keystore configuration

Methods	Description
• `getTrustStoreConfiguration` • `updateTrustStoreConfiguration`	Manage the truststore configuration
• `getHTTPListenerConfigurations` • `addHTTPListenerConfiguration` • `deleteHTTPListenerConfiguration` • `updateHTTPListenerConfiguration`	Manage HTTP listener configurations

Documented in: *Developer Guide: Unwired Server Runtime*, see *Controlling Unwired Server (SUPServer Interface)*

SUPRelayServer API Changes

API changes for the `SUPRelayServer` interface, used to manage the Relay Server or Relay Server cluster for the Unwired Server.

Table 11. New SUPRelayServer Methods

New Methods	Description
• `getOutboundEnablerProxy` • `getOutboundEnablerProxies` • `saveOutboundEnablerProxy` • `updateOutboundEnablerProxy` • `deleteOutboundEnablerProxy` • `deleteOutboundEnablerProxies`	Manage outbound enabler proxy configurations
• `getRelayServer` • `getRelayServers` • `saveRelayServer` • `updateRelayServer` • `deleteRelayServer` • `deleteRelayServers`	Manage relay server configurations

New Methods	Description
• `getServerFarm` • `saveServerFarm` • `updateServerFarm` • `deleteServerFarm` • `deleteServerFarms`	Manage configurations of server farms
• `getServerNode` • `saveServerNode` • `updateServerNode` • `deleteServerNode` • `deleteServerNodes`	Manage configurations of server nodes

Documented in: *Developer Guide: Unwired Server Runtime*, see *Managing Relay Servers*

SUPDomain API Changes

API changes for `SUPDomain`, used to manage domains and their properties.

Table 12. New SUPDomain Methods

Methods	Description
• `exportPackage` • `importPackage`	Manage the importing and exporting of packages
• `getDefaultSecurityConfiguration` • `setDefaultSecurityConfiguration`	Mange the retrieval and setting of the default security configuration for a domain

Documented in: *Developer Guide: Unwired Server Runtime*, *Managing Domains*

SUPMobileWorkflow API Changes

API changes for `SUPMobileWorkflow` , used to administer mobile workflow packages. This API has been deprecated. Use the `SUPMobileHybridApp` API instead.

SUPMobileHybridApp API Changes

API changes for `SUPMobileHybridApp`, used to manage Hybrid Apps and their properties.

Note: `SUPMobileHybridApp` replaces `SUPMobileWorklow`, which has been deprecated.

Table 13. New SUPMobileHybridApp Methods

Methods	Description
• `exportMobileHybridApp` • `importMobileHybridApp`	Manage the importing and exporting of Hybrid Apps
• `setDefaultMobileHybridAppForDevices` • `unsetDefaultMobileHybridAppforDevices` • `getMobileHybridAppCustomIcons`	Manage modifying Hybrid Apps
• `assignMobileHybridAppToTemplates()` • `unassignMobileHybridAppFromTemplates()` • `getTemplateHybridAppAssignments()` • `getTemplateMobileHybridAppStatus()` • `setDefaultMobileHybridAppforTemplates()` • `unsetDefaultMobileHybridAppforTemplates()`	Manage Hybrid App templates
• `getMobileHybridAppClientVariables` • `setMobileHybridAppClientVariables`	Manage client variables for Hybrid Apps

Documented in: *Developer Guide: Unwired Server Runtime*, see *Managing Mobile Workflows*

SUPApplication API Changes

API changes for `SUPApplication`, used to manage applications, application connections, and application connection templates.

Table 14. New SUPApplication Methods

Methods	Description
• `getApplicationPushConfigurations` • `saveApplicationPushConfiguration` • `deleteApplicationPushConfiguration`	Manage native push configurations for an application
• `getApplicationConnectionSettings(Integer numericId, APPCONNECTION_SETTING_FIELD field)`	Manage retrieving the application connection setting for the specified setting field.
• `exportApplication` • `importApplication`	Manage importing and exporting applications

Table 15. Changed SUP Server Configuration Methods

Methods	Description
`createApplicationConnectionTemplate`	Manage creating application connection templates. The Logical Role, Application Identifier, and Security Configuration fields are now validated to make sure they are unique. If they are not unique, the operation fails.

Documented in: *Developer Guide: Unwired Server Runtime*, see *Managing Applications and Application Connections and Templates*

SUPSecurityConfiguration API Changes

API updates for `SUPSecurityConfiguration`, used to define security providers for authentication, authorization, and auditing.

Table 16. New SUPSecurityConfiguration Methods

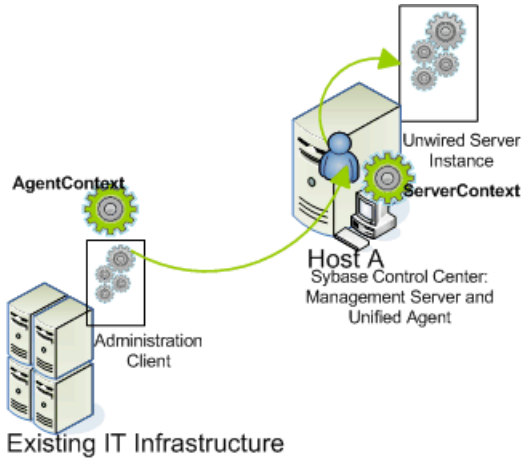
Methods	Description
• `createLogicalRole` • `deleteLogicalRole` • `getRoleMappings` • `updateLogicalRole`	Manage role mappings

Documented in: *Developer Guide: Unwired Server Runtime*, see *Configuring Security Configurations*

Management 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).

Note: Calling deleteXxxx() on a non-existent object causes the API to return silently and not throw an exception.

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`.
`DefaultAdminContext`	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`
`OperationContext`	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 45

Administration Interfaces

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

The Management 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.
`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.
`SUPMobileBusinesObject`	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.

Interface	Includes methods that
`SUPMobileWorkflow`	This interface has been deprecated. Use the `SUPMobileHybridApp` interface instead. Manage and configure deployed mobile workflow packages.
`SUPMobileHybridApp`	Manage and configure deployed Hybrid App 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 268

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 268

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.

Best Practices

Observe these best practices.

- **Thread safety –** The admin API client library is not thread safe, so external synchronization is required when calling the APIs concurrently from multiple threads.
- **Performance –** When managing multiple Sybase Unwired Platform clusters, for performance considerations, it is best to connect to Sybase Control Center co-located with the managed Sybase Unwired Platform cluster. Although connecting to another Sybase Control Center (as long as they share the same credentials) to perform management is supported, performance may not be as good as the former approach.
- **Commit configuration –** The `SUPServerConfiguration` and `SUPSecurityConfiguration` APIs use a local cache and upload changes later to the Unwired Server. You must perform a commit to upload changes to the Unwired Server and then refresh.
- **Error handling –** For error handling, use the error code returned in the exception. Also, by calling the `getErrorCode()` method of `SUPAdminException`, a string representation of the structured error code can be retrieved, which can further the centralized error code handling.

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.
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 *SUP_HOME*\Servers\UnwiredServer\AdminClientAPI\com.sybase.sup.adminapi folder. All other jar files can be found in the *SUP_HOME*\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.

Prerequisites

Ensure the required services are 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.
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="*supPwd*":

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

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",
   "supPwd", 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.

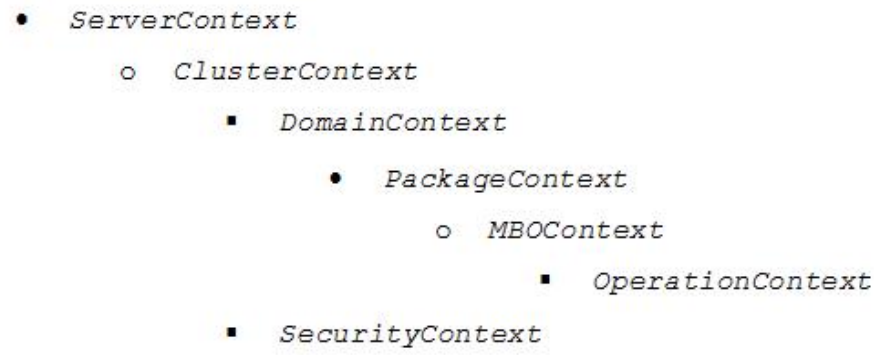
See also

- *Contexts* on page 40

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 javadoc for the Management 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. You can also manage Unwired Server configurations for all nodes in the cluster

Operations you can perform with this interface include:

- Listing member servers, suspending and 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.

The Unwired Server configuration consists of the following components, all of which are metadata-based:

- Communication
  - Administration Listener
  - HTTP / HTTPS Listener
  - SSL Security Profile
  - Key Store
  - Trust Store
- Messaging
- Replication

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: Unwired Server Runtime > 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);
```

Retrieve HTTP Proxy Setting

Retrieve the HTTP proxy setting.

Syntax

```
HttpProxySettingVO getHttpProxySetting()
      throws SUPAdminException;
```

Returns

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

Examples

- **Get Http Proxy Setting**

```
HttpProxySettingVO hp = supCluster.getHttpProxySetting();
System.out.println(hp.getMessagingProxyMaxConnectionsPerHost());
System.out.println(hp.getMessagingProxyThreadCounts());
System.out.println(hp.getReplicationProxyThreadCounts());
```

Update HTTP Proxy Setting

Update the HTTP proxy setting.

Syntax

```
void updateHttpProxySetting(HttpProxySettingVO proxySetting)
      throws SUPAdminException;
```

Parameters

- **proxySetting –** The proxy setting that you want to update.

Returns

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

Examples

- **Update Http Proxy Setting**

```
HttpProxySettingVO hp = new HttpProxySettingVO();
hp.setMessagingProxyThreadCounts(20);
hp.setReplicationProxyThreadCounts(17);
hp.setMessagingProxyMaxConnectionsPerHost(3000);
supCluster.updateHttpProxySetting(hp);
```

Retrieve Cluster Context for Web Container Configuration

Retrieve the cluster configuration from server and cache it locally. Only a Sybase Unwired Platform administrator or the Sybase Unwired Platform help desk can perform this.

This refreshes the cluster configurations, which includes following components: Management, Replication, Messaging, SolutionManager, RelayServer, SecurityProfile, KeyStore, TrustStore, OCSP, and Performance. The returned `ConfigurationValidationStatus` contains the validation status of the cluster configuration at server side.

Syntax

```
ConfigurationValidationStatus refreshClusterConfiguration()
      throws SUPAdminException;
```

Returns

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

Commit Server Changes in Local Cache Refresh

Deliver the local server configuration to the server to be saved. Only a Sybase Unwired Platform administrator can perform this.

All the metadata based configurations are delivered, including cluster configuration, Web container configuration, and configuration cache. The returned `ConfigurationValidationStatus` contains the validation status of the delivered Sybase Unwired Platform configuration at server side.

Syntax

```
ConfigurationValidationStatus commit()
      throws SUPAdminException;
```

Returns

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

Refresh Server Changes from Local Cache

Retrieve all Sybase Unwired Platform configurations from server and cache it locally. Only a Sybase Unwired Platform administrator or the Sybase Unwired Platform help desk can perform this.

This refreshes all meta-data based configurations, including: cluster configuration, web container configuration, and configuration cache. The returned `ConfigurationValidationStatus` contains the validation status of the Sybase Unwired Platform configuration at server side.

Syntax

```
ConfigurationValidationStatus refresh()
      throws SUPAdminException;
```

Returns

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

Retrieve Server Context for Cluster

Retrieve the server context associated with this cluster configuration.

Syntax

```
ClusterContext getContext();
```

Returns

Returns the server context.

Examples

- –

Retrieve Management Configuration

Retrieve the management related configuration.

Syntax

```
SUPConfigurationComponentVO getManagementConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve Management configuration**

```
SUPConfiguration supConf = SUPObjectFactory
        .getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf

        .getManagementConfiguration();
System.out.println(confVO.getProperties);
```

Update Management Configurations

Update the properties of the management related configuration.

Syntax

```
void updateManagementConfiguration(SUPConfigurationComponentVO
configurationComponent)
      throws SUPAdminException;
```

Parameters

- **portVO –**

Returns

If successful, returns silently. If unsuccessful, throws SUPAdminException

Examples

- **Update Management Configuration**

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
        .getManagementConfiguration();
confVO.getProperties().put("sup.admin.port", "2003");
supConf.updateManagementConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve Security Profile Configuration

Retrieve a list of SSL security profile configurations.

Syntax

```
Collection<SUPConfigurationComponentVO>
getSSLSecurityProfileConfigurations()
     throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve Security Profiles**

```
SUPConfiguration supConf = SUPObjectFactory
          .getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
for(SUPConfigurationComponentVO scvo :
supConf.getSSLSecurityProfileConfigurations()){
   System.out.println(scvo.getID());
   System.out.println(scvo.getType());
   System.out.println(scvo.getProperties());
}
```

Add Security Profile Configuration

Add a new SSL security profile configuration.

Syntax

```
 void addSSLSecurityProfileConfiguration(SUPConfigurationComponentVO
serverComponent)
      throws SUPAdminException;
```

Parameters

- **serverComponent –**

Returns

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

Examples

-

Delete Security Profile Configuration

Delete a SSL security profile configuration.

Syntax

```
void deleteSSLSecurityProfileConfiguration(String serverComponentID)
      throws SUPAdminException;
```

Parameters

- **serverComponentID –** The ID of the security profile configuration you want to delete.

Returns

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

Examples

- –

Update Security Profile Configuration

Update a SSL security profile configuration.

Syntax

```
void
updateSSLSecurityProfileConfiguration(SUPConfigurationComponentVO
serverComponent)
      throws SUPAdminException;
```

Parameters

- **serverComponent –** The server component that you want to update.

Returns

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

Examples

- **Update Security Profile Configuration**

  ```
  supConf.refreshClusterConfiguration();
  SUPConfigurationComponentVO scvo = supConf
          .getSSLSecurityProfileConfigurations().iterator().next();
  scvo.getProperties().put("certificateLabel", "mycertificate");
  supConf.updateSSLSecurityProfileConfiguration(scvo);
  supConf.commitClusterConfiguration();
  ```

Retrieve Keystore Configuration

Retrieve the properties of the key store configuration.

Syntax

```
SUPConfigurationComponentVO getKeyStoreConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve**

```
 SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getKeyStoreConfiguration();
System.out.println(confVO.getProperties);
```

Update Keystore Configuration

Update the key store configuration.

Syntax

```
void updateKeyStoreConfiguration(SUPConfigurationComponentVO
serverComponent)
      throws SUPAdminException;
```

Parameters

- **serverComponent –** The server component you want to update.

Returns

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

Examples

- **Update Keystore Configuration**

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
        .getKeyStoreConfiguration();
confVO.getProperties().put("sup.sync.sslkeystore_password", "
changeit");
```

```
supConf.updateKeyStoreConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve Trust Store Configuration

Retrieve the properties of the trust store configuration.

Syntax

```
SUPConfigurationComponentVO getTrustStoreConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Update TrustStore Configuration**

```
SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getTrustStoreConfiguration();
System.out.println(confVO.getProperties);
```

Update Trust Store Configuration

Update the trust store configuration.

Syntax

```
void updateTrustStoreConfiguration(SUPConfigurationComponentVO
serverComponent)
      throws SUPAdminException;
```

Parameters

- **serverComponent –** The server component that you want to update.

Returns

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

Examples

- **Update TrustStore Configuration**

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
        .getTrustStoreConfiguration();
```

```
confVO.getProperties().put("sup.sync.sslkeystore_password", "
changeit");
supConf.updateTrustStoreConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Enable/Disable OCSP Configuration

Enable or disable OCSP configuration.

Syntax

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

Parameters

- **flag –** Set the flag to TRUE to enable and to FALSE to disable.

Returns

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

Examples

- **Enable OCSP Configuration**

```
SUPConfiguration supConf = SUPObjectFactory
          .getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
supConf.enableOCSPConfiguration(true);
supConf.commitClusterConfiguration();
```

Retrieve OCSP Configuration

Retrieve the OCSP configuration. Only a Sybase Unwired Platform administrator or the Sybase Unwired Platform help desk can perform this.

Syntax

```
SUPConfigurationComponentVO getOCSPConfiguration() throws
SUPAdminException;
```

Returns

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

Examples

- **Retrieve OCSP Configuration**

```
SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
```

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getOCSPConfiguration();
System.out.println(confVO.getProperties);
```

Update OCSP Configuration

Update the OCSP configuration. Only a Sybase Unwired Platform administrator can perform this.

Syntax

```
void updateOCSPConfiguration(SUPConfigurationComponentVO
serverComponent)
      throws SUPAdminException;
```

Parameters

- **serverComponent –** The server component that you want to update.

Returns

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

Examples

- **Update OCSP Configuration**

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getOCSPConfiguration();
confVO.getProperties().put("ocsp.responderURL", "http://
apple.com");
supConf.updateOCSPConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve Performance Configuration

Retrieve the performance configuration.

Syntax

```
SUPConfigurationComponentVO getPerformanceConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve Performance Configuration**

```
 SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getPerformanceConfiguration();
System.out.println(confVO.getProperties);
```

Update Performance Configuration

Update the properties of the performance configuration.

Syntax

```
void updatePerformanceConfiguration(SUPConfigurationComponentVO
serverComponent)
      throws SUPAdminException;
```

Parameters

- **serverComponent –** The sever component that you want to update.

Returns

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

Examples

- **Update Performance Configuration**

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getPerformanceConfiguration();
confVO.getProperties().put("sup.msg.inbound_count", "70");
supConf.updatePerformanceConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve Cache Configuration

Retrieve the distribution configuration. Only a Sybase Unwired Platform administrator and Sybase Unwired Platform help desk can perform this.

Syntax

```
 SUPConfigurationComponentVO getConfigurationCache()
      throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve Configuration Cache**

```
SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
supConf.refreshConfigurationCache();
SUPConfigurationComponentVO confVO = supConf
      .getConfigurationCache();
System.out.println(confVO.getProperties);
```

Update Cache Configuration

Update the properties of the cache configuration. Only a Sybase Unwired Platform administrator can do this.

Syntax

```
 void updateConfigurationCache(SUPConfigurationComponentVO
serverComponent)
      throws SUPAdminException;
```

Parameters

- **serverComponentID –**

  The ID of the server component you want to update.
- **serverComponent –**

  The server component you want to update.

Returns

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

Examples

- **Update Configuration Cache**

```
 supConf.refreshConfigurationCache();
SUPConfigurationComponentVO confVO = supConf
        .getConfigurationCache();
confVO.getProperties().put("cache.core.pool.size", "100");
supConf.updateConfigurationCache(confVO);
supConf.commitConfigurationCache();
```

Retrieve Web Container Configuration

Retrieve the common web container configuration.

Syntax

```
SUPConfigurationComponentVO getWebContainerCommonConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve Common WebContainer Configuration**

```
SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
supConf. refreshWebContainerConfiguration();
SUPConfigurationComponentVO confVO = supConf

      . getWebContainerCommonConfiguration();
System.out.println(confVO.getProperties);
```

Update Web Container Configuration

Update the properties of the administration listener configuration.

Syntax

```
void
updateWebContainerCommonConfiguration(SUPConfigurationComponentVO
webContainerComponent)
      throws SUPAdminException;
```

Returns

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

Examples

- **Update Common WebContainer Configuration**

```
supConf.refreshWebContainerConfiguration();
SUPConfigurationComponentVO confVO = supConf

      .getWebContainerCommonConfiguration();
confVO.getProperties().put("maxFormContentSize", "2500000");
confVO.getProperties().put("gzipFilter", "true");
supConf.updateWebContainerCommonConfiguratio(confVO);
supConf.commitWebContainerConfiguration();
```

Update Messaging Configuration

Update the properties of the messaging related configuration.

Syntax

```
void updateMessagingConfiguration(SUPConfigurationComponentVO
configurationComponent)
      throws SUPAdminException;
```

Parameters

- **configurationComponent –** The messaging component configuration properties to update to.

Returns

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

Examples

- **Update Messaging Configuration**

```
SUPConfigurationComponentVO confVO =
supConf.getMessagingConfiguration();
confVO.getProperties().put("msg.http.server.proxy.ports",
"5002");
supConf.updateMessagingConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve Messaging Configuration

Retrieve the messaging related configuration.

Syntax

```
SUPConfigurationComponentVO getMessagingConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Get Messaging Configuration**

```
SUPConfiguration supConf =
SUPObjectFactory.getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO =
supConf.getMessagingConfiguration();
System.out.println(confVO.getProperties);
```

Update Replication Configuration

Update the properties of the replication related configuration.

Syntax

```
void updateReplicationConfiguration(SUPConfigurationComponentVO
configurationComponent)
      throws SUPAdminException;
```

Parameters

- **configurationComponent –** The replication configuration component properties to update to.

Returns

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

Examples

- **Update SUP Replication Configuration**

```
SUPConfigurationComponentVO confVO =
supConf.getReplicationConfiguration();
confVO.getProperties().put("sup.sync.port", "2480");
confVO.getProperties().put("sup.sync.httpsport", "2481");
confVO.getProperties().put("sup.sync.protocol", "http,https");
confVO.getProperties().put("sup.sync.e2ee_type", "RSA");
confVO.getProperties().put("sup.sync.e2ee_private_key",
"Repository/Certificate/e2ee_private_key.key");
confVO.getProperties().put("sup.sync.e2ee_private_key_password",
"sybase1");
confVO.getProperties().put("sup.sync.e2ee_public_key",
"Repository/Certificate/e2ee_public_key.key");
confVO.getProperties().put("sup.sync.certificate", "Repository/
Certificate/https_server_identity.crt");
confVO.getProperties().put("sup.sync.certificate_password",
"sybase1");
confVO.getProperties().put("sup.sync.public.certificate",
"Repository/Certificate/https_public_cert.crt");
confVO.getProperties().put("relayserver.trusted_certs", "");
confVO.getProperties().put("sup.user.options", "-zf");
supConf.updateMessagingConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve Replication Configuration

Retrieve the replication related configuration.

Syntax

```
SUPConfigurationComponentVO getReplicationConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve SUP Replication Configuration**

```
SUPConfiguration supConf =
SUPObjectFactory.getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO =
supConf.getReplicationConfiguration();
System.out.println(confVO.getProperties);
```

Retrieve HTTP Listener Configuration

Retrieve a list of HTTP listener configurations.

Syntax

```
Collection<SUPCOnfigurationComponentVO>
getHTTPListenerConfigurations()
      throws SUPAdminException;
```

Returns

If successful, returns a collection of HTTP listener configurations. If unsuccessful, throws SUPAdminException.

Examples

- **Retrieve HTTP Listener Configuration**

```
SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
supConf.refreshWebContainerConfiguration();
for(SUPConfigurationComponentVO scvo :
supConf.getHTTPListenerConfigurations()){
   System.out.println(scvo.getID());
   System.out.println(scvo.getType());
```

```
   System.out.println(scvo.getProperties());
}
```

Add HTTP Listener Configuration

Add a new HTTP listener configuration.

Syntax

```
void addHTTPListenerConfiguration(SUPConfigurationComponentVO
webContainerComponent)
      throws SUPAdminException;
```

Returns

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

Examples

- **Add HTTP Listener Configuration**

```
supConf.refreshWebContainerConfiguration();
SUPConfigurationComponentVO confVO = supConf

        .getHTTPListenerConfigurations().iterator().next();
confVO.getProperties().put("port", "8100");
supConf.addHTTPListenerConfiguration(confVO);
supConf.commitWebContainerConfiguration();
```

Delete HTTP Listener Configuration

Delete a HTTP listener configuration .

Syntax

```
 void deleteHTTPListenerConfiguration(String
httpListenerComponentID)
      throws SUPAdminException;
```

Returns

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

Examples

- **Delete HTTP Listener Configuration**

```
supConf.refreshWebContainerConfiguration();
SUPConfigurationComponentVO confVO = supConf
        .getHTTPListenerConfigurations().iterator().next();
supConf.deleteHTTPListenerConfiguration(confVO.getID());
supConf.commitWebContainerConfiguration();
```

Update HTTP Listener Configuration

Update a HTTP listener configuration.

Syntax

```
void updateHTTPListenerConfiguration(SUPConfigurationComponentVO
webContainerComponent)
      throws SUPAdminException;
```

Returns

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

Examples

- **Update HTTP Listener Configuration**

```
supConf.refreshWebContainerConfiguration();
SUPConfigurationComponentVO confVO = supConf
        .getHTTPListenerConfigurations().iterator().next();
confVO.getProperties().put("maxThreads", "50");
supConf.updateHTTPListenerConfiguration(confVO);
supConf.commitWebContainerConfiguration();
```

Retrieve Solution Manager Configuration

Retrieve the solution manager configuration.

Syntax

```
SUPConfigurationComponentVO getSolutionManagerConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve Solution Manager Configuration**

```
SUPConfiguration supConf = SUPObjectFactory
      .getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getSolutionManagerConfiguration();
System.out.println(confVO.getProperties);
```

Update Solution Manager Configuration

Update the properties of the solution manager configuration.

Syntax

```
void updateSolutionManagerConfiguration(SUPConfigurationComponentVO
configurationComponent)
      throws SUPAdminException;
```

Parameters

- **configurationComponent –** The solution manager configuration properties to upgrade to.

Returns

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

Examples

- **Update Solution Manager Configuration**

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO = supConf
      .getSolutionManagerConfiguration();
confVO.getProperties().put("com.sap.solutionmanager.url",
"http://solutionManagerHost");
supConf.updateSolutionManagerConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve DCN Configuration

Retrieve the DCN Configuration.

Syntax

```
SUPConfigurationComponentVO getDCNConfiguration()
      throws SUPAdminException;
```

Returns

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

Examples

- **Get DCN Configuration**

```
SUPConfiguration supConf =
SUPObjectFactory.getSUPConfiguration(clusterContext);
supConf.refreshClusterConfiguration();
```

```
SUPConfigurationComponentVO confVO =
supConf.getDCNConfiguration();
System.out.println(confVO.getProperties);
```

Update DCN Configuration

Update the DCN configuration properties.

Syntax

```
void updateDCNConfiguration(SUPConfigurationComponentVO
configurationComponent)
      throws SUPAdminException;
```

Parameters

- **configurationComponent –** The DCN configuration properties to update to.

Returns

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

Examples

- **Update DCN Configuration**

```
supConf.refreshClusterConfiguration();
SUPConfigurationComponentVO confVO =
supConf.getDCNConfiguration();
confVO.getProperties().put("sup.dcn.http.get.enabled","true");
supConf.updateDCNConfiguration(confVO);
supConf.commitClusterConfiguration();
```

Retrieve Server Log Setting

Retrieve the server log settings.

Syntax

```
ServerLogSettingVO getServerLogSetting()
      throws SUPAdminException;
```

Returns

If successful, returns the specified type (can be null). If unsuccessful, throws SUPAdminException.

Examples

- **Get Server Log Setting**

```
SUPCluster supCluster =
SUPObjectFactory.getSUPCluster(clusterContext);
```

```
ServerLogSettingVO sls = supCluster.getServerLogSetting();
System.out.println(sls);
```

Update Server Log Setting

Update the server log settings.

Syntax

```
void updateServerLogSetting(ServerLogSettingVO setting)
      throws SUPAdminException;
```

Parameters

- **setting –** The server log setting to update.

Returns

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

Examples

- **Update Server Log Setting**

```
ServerLogSettingVO sls = supCluster.getServerLogSetting();
sls.setMaxBackupLogs(15);
sls.setMaxLogFileSize("20mb");
sls.setNewLogWhenServerStarted(false);
Collection<ServerLogComponentLogLevel> loglevels =
sls.getComponentLogLevels();
for(ServerLogComponentLogLevel level : loglevels){
      if(level.getComponent().equals(SERVER_LOG_COMPONENT.Other))
{
            level.setLoglevel(LOG_LEVEL.WARN);
            break;
      }
}
supCluster.updateServerLogSetting(sls);
```

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*.

Retrieval and Setting of Trace Configuration

Retrieves and sets the trace configuration settings.

Syntax

```
Collection<TraceConfigVO> getTraceConfigs() throws
SUPAdminException;

void setTraceConfigs(configs) throws SUPAdminException;
```

Returns

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

Examples

- **Retrieve and set trace configuration settings**

```
Collection<TraceConfigVO> configs = supCluster.getTraceConfigs();
for (TraceConfigVO config : configs) {
        if
(TRACE_LOG_MODULE.JMS_BRIDGE.equals(config.getModule())) {
            config.setLevel(TRACE_LOG_LEVEL.DEBUG);
        }
}
supCluster.setTraceConfigs(configs);
System.out.println(configs);
```

Setting Time Zone

When the time zone of the administration client is different from that of the Unwired Server, you must format the time zone.

- If a time or date string representation is returned to the client, it must be formatted using the Unwired Server's time zone. This requires the API implementation to perform the formatting; the client is not required to perform it.
- If a time or date string representation is passed to the API, it must be formatted in the Unwired Server's time zone. This requires the client to perform the formatting before passing the time or date to API.

- If a time or date is of `java.util.Date`, `java.util.Calendar`, or `java.sql.Timestamp`, it can be used as it is.

Syntax

```
TimeZone getTimeZone throws SUPAdminException;

void setTimeZone(timezone) throws SUPAdminException;
```

Returns

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

Examples

- **Setting the Time Zone –** This example shows how to meet the time zone requirements.

```
TimeZone tz = supCluster.getTimeZone();
ClusterContext clusterContext = supCluster.getContext();
clusterContext.setTimeZone(tz);
DomainContext domainContext =
clusterContext.getDomainContext("<domain name>");
```

Usage

Execute these methods before making any timezone related API calls.

SAP License Audit

For SAP® built applications, generate an XML file that contains usage audit data that is then uploaded to SAP License Audit.

Syntax

```
String auditMeasurement =
    supCluster.generateSAPAuditMeasurement(userName);
```

Returns

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

Examples

- **Generate audit measurement file: –**

```
String auditMeasurement =
supCluster.generateSAPAuditMeasurement(“John Doe”);
```

Retrieve BES Response Portioning Limit

Retrieves the BES response portioning value for a cluster.

There are limits on the amount of data that can be downloaded through an HTTP connection using the BlackBerry MDS. The BES response portioning limit determines the amount of

HTTP traffic the BES MDS server accepts from Unwired Server. For BlackBerry MDS, this limit is set in BlackBerry Manager using the Maximum KB/Connection setting.

Syntax

```
getBESResponsePortioningLimit() throws SUPAdminException
```

Returns

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

Examples

- **Retrieve BES portioning limit –** retrieves the BES reponse portioning limit for a cluster

  ```
  Review BES response portioning:
  Integer limit = supCluster. getBESResponsePortioningLimit ();
  System.out.println(limit);
  ```

Update BES Response Portioning Limit

Updates the BES response portioning limit for a cluster.

There are limits on the amount of data that can be downloaded through an HTTP connection using the BlackBerry MDS. The BES response portioning limit determines the amount of HTTP traffic the BES MDS server accepts from Unwired Server. For BlackBerry MDS, this limit is set in BlackBerry Manager using the Maximum KB/Connection setting.

Syntax

```
void setBESResponsePortioningLimit(Integer limit) throws
SUPAdminException;
```

Parameters

- **limit –** The BES response portioning limit value.

Returns

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

Examples

- **Update BES response portioning limit –** Update the BES response portioning limit to 3 bytes.

  ```
  Update BES response portioning:
  SUPCluster. setBESResponsePortioningLimit(3);
  ```

Managing Relay Servers

The `SUPRelayServer` interface allows you to manage the relay server or relay server cluster for the Unwired Server.

Operations you can perform with this interface include:

- Starting an administration session for a relay server.
- Creating, updating and deleting a relay server configuration.
- Creating, updating and deleting a relay server farm configuration.
- Creating, updating and deleting relay server node information.

Start Relay Server Management

Starts the management of a relay server.

Syntax

```
SUPRelayServer getSUPRelayServer(ClusterContext clusterContext)
      throws SUPAdminException;
```

Parameters

- **clusterContext –**

  The specified relay server to manage.

Returns

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

Examples

- **Relay server startup –** starts the management of the specified relay server.

  ```
  SUPRelayServer suprs =
  SUPObjectFactory.getSUPRelayServer(contextFactory
                  .getClusterContext());
  ```

Usage

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

Retrieve Relay Server Configuration

Retrieves one or more relay server configurations.

You can retrieve a list of relay server configurations, or a specific relay server configuration based on a given ID, or a given host name and port number.

Syntax

```
java.util.List<RelayServerVO> getRelayServers()
                                                  throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
RelayServerVO getRelayServer(java.lang.Integer relayServerId)
                             throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
RelayServerVO getRelayServer(java.lang.String host,
                             java.lang.Integer port)
                             throws
com.sybase.sup.admin.exception.SUPAdminException;
```

Parameters

- **relayServerId –** The ID of the relay server configuration you want to retrieve.
- **host –** The host name of the relay server configuration you want to retrieve.
- **port –** The port of the relay server configuration you want to retrieve.

Returns

If successful, retrieves a list of relay server configurations, or a specific relay server configuration. If unsuccessful, returns SUPAdminException.

Examples

- **Retrieval of a list of relay servers –** retrieves a list of all relay server configurations.

  ```
  Iterator<RelayServerVO> iter =
  suprs.getRelayServers().iterator();
          if (!iter.hasNext())
              return;
          RelayServerVO rsvo = iter.next();
          int id = rsvo.getID();
  ```

- **Retrieval of a relay server with a given ID –** retrieves the relay server configuration with the given ID.

  ```
  RelayServerVO rsvo_byId = suprs.getRelayServer(id);
  ```

- **Retrieval of a relay server with a given host and port –** retrieves the relay server configuration with the given host name and port.

  ```
  RelayServerVO rsvo_hostPort =
  suprs.getRelayServer(rsvo.getHost(),
                  rsvo.getPort());
  ```

Create Relay Server Configuration

Creates a relay server configuration.

Syntax

```
saveRelayServer(RelayServerVO relayServer) throws
SUPAdminException;
```

Parameters

- **relayServer –** The relay server configuration to be created.

Returns

If successful, creates a new relay server configuration. If unsuccessful, returns SUPAdminException.

Examples

- **Creation of a relay server –** Creates a relay server configuration with server credentials, server farms, and server nodes.

```
RelayServerVO rsvo = new RelayServerVO();

RelayServerVOBuilder bld = new RelayServerVOBuilder(rsvo);
        bld.host("myrelayserver.sybase.com").
            port(1234).
              credential("supAdmin").password("s3pAdmin").back().

credential("supAdmin2").password("s3pAdmin2").back().
                deleteFarm("supAdmin2.SupAdminAutoTestRBS").
                farm("supAdmin2.SupAdminAutoTestMBS").
                  description("Descrition from API - Messaging").
                    type(SERVER_FARM_TYPE.MESSAGING).
                    name("supAdmin2.SupAdminAutoTestMBS - API").
                    deleteServerNode("node-name").
                    serverNode("node1").
                        token("node1-token").
                        back().
                    back().
                farm("farm - API").
                 description("Descrition from API - Replication").
                    type(SERVER_FARM_TYPE.REPLICATION).
                    serverNode("node2").
                        token("node2-token").
                        back().
                    back().
                back();

        rsvo = bld.build();
        suprs.saveRelayServer(rsvo);
```

Update Relay Server Configuration

Updates an existing relay server configuration with a given ID.

Syntax

```
updateRelayServer(java.lang.Integer relayServerId,
                        RelayServerVO relayServer) throws
SUPAdminException;
```

Parameters

- **relayServerID –** The new ID of the relay server configuration.
- **relayServer –** The relay server configuration to be updated.

Returns

If successful, updates the relay server configuration. If unsuccessful, returns SUPAdminException.

Examples

- **Update of a relay server configuration –** Updates the relay server with the given ID.

```
@Test
public void updateRelayServer() throws Exception {
    List<RelayServerVO> relayServers =
supRelayServer.getRelayServers();
    if (relayServers.size() == 0) {
        System.out.println("No relay server defined.");
        return;
    }
    RelayServerVO relayServer = relayServers.get(0);
    relayServer.setHost("myRelayServer2.sybase.com");
    relayServer.setPort(8080);
    relayServer
          .setDescription("The relay server host and port has been
updated.");
    supRelayServer.updateRelayServer(relayServer.getID(),
relayServer);
}
```

Delete Relay Server Configurations

Deletes one or more relay server configurations.

You can delete the following:

- a group of relay servers based on a list
- a group of relay servers based on a list of IDs
- a specific relay server based on an ID

- a specific relay server based on a host name and port number

Syntax

```
void deleteRelayServers(java.util.List<RelayServerVO> relayServers)
                        throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
void deleteRelayServers(java.util.List<RelayServerVO> relayServers)
                        throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
deleteRelayServers(java.util.Set<java.lang.Integer> relayServerIds)
                        throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
deleteRelayServer(java.lang.Integer relayServerId)
                       throws
com.sybase.sup.admin.exception.SUPAdminException;
```

Parameters

- **relayServers –** The list of relay server configurations that you want to delete.
- **relayServerIds –** The list of relay server configuration IDs that you want to delete.
- **relayServerId –** The ID of the relay server configuration you want to delete.
- **host –** The host name of the relay server configuration you want to delete.
- **port –** The port of the relay server configuration you want to delete.

Returns

If successful, deletes a list of relay servers, or a specific relay server. If unsuccessful, returns SUPAdminException.

Examples

- **Deletion of list of relay servers –** deletes all relay server configurations in the list.

  ```
  suprs.deleteRelayServers(suprs.getRelayServers());
  ```

- **Deletion of list of relay servers by ID –** deletes all relay server configurations with the given IDs.

  ```
  Set<Integer> relayServerIDsToDelete = new HashSet<Integer>();
  relayServerIDsToDelete.add(1);
  relayServerIDsToDelete.add(2);
  supRelayServer.deleteRelayServers(relayServerIDsToDelete);
  ```

- **Deletion of a relay server with a given ID –** deletes the relay server configuration with the given ID.

  ```
  List<RelayServerVO> rss = suprs.getRelayServers();
          Iterator<RelayServerVO> iter = rss.iterator();
          suprs.deleteRelayServer(iter.next().getID());
  ```

- **Deletion of a relay server with a given host and port –** deletes the relay server with the given host name and port.

```
supRelayServer.deleteRelayServer("myRelayServer.sybase.com", 80);
```

Retrieve Server Node

Retrieve a server node configuration wtih the given ID.

Syntax

```
ServerNodeVO getServerNode(Integer serverNodeId)
      throws SUPAdminException;
```

Parameters

- **serverNodeId –** The ID of the server node configuration.

Returns

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

Examples

- **Retrieve Server Node –** Get the server node with a primary key of 1.

```
@Test
    public void testGetServerNode() throws Exception {
        ServerNodeVO serverNode = suprs.getServerNode(1);
        if (serverNode != null) {
            String serverNodeName = serverNode.getName();
            System.out.println(serverNodeName);
        }
    }
```

Create Server Node

Create a server node configuration if there is none. Otherwise, the existing one will be updated.

Syntax

```
void saveServerNode(ServerNodeVO serverNode)
      throws SUPAdminException;
```

Parameters

- **serverNode –** The server configuration to be created or updated.

Returns

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

Examples

- **Add Server Node –** Add a server node into a farm with a primary key of 1 in the database.

```
@Test
    public void testSaveServerNode() throws Exception {
        ServerNodeVO serverNode = new ServerNodeVO();
        serverNode.setName("node1");
        serverNode.setToken("myOuboundEnablerToken");
       ServerFarmVO serverFarm = supRelayServer.getServerFarm(1);
        serverNode.setServerFarm(serverFarm);
        supRelayServer.saveServerNode(serverNode);
    }
```

Delete Server Node

Delete a set of server node configurations with the given IDs or a single server node with the given ID.

Syntax

```
void deleteServerNodes(java.util.Set<Integer> serverNodeIds)
      throws SUPAdminException;
void deleteServerNode(Integer serverNodeId)
      throws SUPAdminException;
```

Parameters

- **serverNodeIds –** A set of server node configuration IDs.
- **serverNodeId –** A server node configuration ID.

Returns

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

Examples

- **Delete a Server Node**

```
@org.junit.Test
public void testDeleteServerNode() throws Exception {
    RelayServerVO relayServer = supRelayServer.getRelayServer(
            "myrelayserver.sybase.com", 80);
    if (relayServer == null) {
        System.out
                .println("myrelayserver.sybase.com:80 does not
configured");
        return;
        }
    List<ServerFarmVO> farms = relayServer.getServerFarms();
    if (farms.size() == 0) {
        System.out.println("No farm defined.");
        return;
    }
```

```
    ServerFarmVO farm = farms.get(0);
    List<ServerNodeVO> nodes = farm.getServerNodes();
    if (nodes.size() == 0) {
        System.out.println("No server node defined in farm "
                + farm.getName());
        return;
    }
    ServerNodeVO nodeToDelete = nodes.get(0);
    supRelayServer.deleteServerNode(nodeToDelete.getID());
}
```

Update Server Node

Update an existing server node configuration with the given ID.

Syntax

```
void updateServerNode(java.lang.Integer serverNodeId,
ServerNodeVO serverNode)
      throws SUPAdminException;
```

Parameters

- **serverNodeId –** The server node configuration ID.
- **serverNode –** The value object containing the property values to be updated.

Returns

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

Examples

- **Update Server Node –** Update the name and token value of the server node whose ID is 1.

```
@org.junit.Test
public void testUpdateServerNode() throws Exception {
        ServerNodeVO node = supRelayServer.getServerNode(1);
        if (node == null) {
            System.out.println("There is no server node with ID
1.");
        }
        node.setName("node1_upated");
        node.setToken("token_updated");
        supRelayServer.updateServerNode(node.getID(), node);
    }
```

Retrieve Server Farm Configuration

Retrieve the server farms that belong to this relay server.

Syntax

```
ServerFarmVO getServerFarm(Integer serverFarmID)
      throws SUPAdminException;
```

Parameters

- **serverFarmId –** The ID of the server farm you want to retrieve.

Returns

If successful, returns a list of server farms. If unsuccessful, throws SUPAdminException.

Examples

- **Retrieve Server Farm –** Get server farm with a primary key of 1 in the database.

```
@org.junit.Test
    public void testGetServerFarm() throws Exception {
        ServerFarmVO serverFarm = supRelayServer.getServerFarm(1);
        if (serverFarm != null) {
            String farmName = serverFarm.getName();
            System.out.println(farmName);
            SERVER_FARM_TYPE type = serverFarm.getType();
            System.out.println(type);
        }
    }
```

Create Server Farm Configuration

Creates a server farm configuraiton if none exists. Otherwise, the existing one is updated.

Syntax

```
void saveServerFarm(ServerFarmVO serverFarm)
      throws SUPAdminException;
```

Parameters

- **serverFarm –** The server farm configuration to be created or updated..

Returns

If successful, returns silently. If unsuccessful, throws SUPAdminException

Examples

- **Add a Server Farm –** Add a new farm into the relay server whose ID is 1.

```
@org.junit.Test
    public void testSaveServerFarm() throws Exception {
        ServerFarmVO serverFarm = new ServerFarmVO();
        serverFarm.setName("messagingFarm1");
        serverFarm.setType(SERVER_FARM_TYPE.MESSAGING);
        serverFarm.setDescription("This is a sample messaging
farm.");
        RelayServerVO relayServer =
supRelayServer.getRelayServer(1);
        if (relayServer == null) {
            System.out
                    .println("There is no relay server with ID 1.
Cannot add a farm into this relay server.");
            return;
        }
        serverFarm.setRelayServer(relayServer);
        supRelayServer.saveServerFarm(serverFarm);
    }
```

Update Server Farm Configuration

Update an existing server farm configuration with the given ID.

Syntax

```
void updateServerFarm(Integer serverFarmId, ServerFarmVO serverFarm)
      throws SUPAdminException;
```

Parameters

- **serverFarmId –** The server farm configuration ID.
- **serverFarm –** The value object containing the property values to be updated.

Returns

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

Examples

- **Retrieve and Update Farm Configuration –** Retrieve a farm and update the name and type values.

```
@org.junit.Test
public void testUpdateServerFarm() throws Exception {
        List<RelayServerVO> relayServers =
supRelayServer.getRelayServers();
        if (relayServers.size() == 0) {
            System.out.println("The SUP cluster do not have any
relay server configured.");
            return;
```

```
        }
        RelayServerVO relayServer = relayServers.get(0);
        List<ServerFarmVO> farms = relayServer.getServerFarms();
        if (farms.size() == 0) {
            System.out.println("Relay server " +
relayServer.getHost() + ":"
                    + relayServer.getPort() + " do not have any
farm.");
            return;
        }
        ServerFarmVO farm = farms.get(0);
        farm.setName("farm_name_updated");
        farm.setType(SERVER_FARM_TYPE.WEBSERVICE);
       farm.setDescription("This is a farm that name and type has
been updated.");
        supRelayServer.updateServerFarm(farm.getID(), farm);
    }
```

Delete Server Farm Configuration

Delete a server farm configuration or a set of server farm configurations with the given ID or set of IDs.

Syntax

```
void deleteServerFarm(Integer serverFarmId)
     throws SUPAdminException;
void deleteServerFarms(java.util.Set<Integer> serverFarmIds)
     throws SUPAdminException;
```

Parameters

- **serverFarmId –** A server farm configuration ID you want to delete.
- **serverFarmIds –** The set of server farm configuration IDs you want to delete.

Returns

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

Examples

- **Delete a Farm**

```
@org.junit.Test
public void testDeleteFarm() throws Exception {
    RelayServerVO relayServer = supRelayServer.getRelayServer(
            "myrelayserver.sybase.com", 80);
    if (relayServer == null) {
        System.out
                .println("myrelayserver.sybase.com:80 does not
configured");
        return;
    }
    List<ServerFarmVO> farms = relayServer.getServerFarms();
```

```
    if (farms.size() == 0) {
        System.out.println("No farm defined.");
        return;
    }
    ServerFarmVO farm = farms.get(0);
    supRelayServer.deleteServerFarm(farm.getID());
}
```

Retrieve Outbound Enabler Proxy Configuration

Retrieves one or more outbound enabler proxy configurations.

You can retrieve a list of outbound enabler proxy configurations, or a specific outbound enabler proxy configuration based on a given ID, or a given host name and port number.

Syntax

```
java.util.List<OutboundEnablerProxyVO> getOutboundEnablerProxies()
                                                                    throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
OutboundEnablerProxyVO getOutboundEnablerProxy(java.lang.Integer
outboundEnablerProxyId)
                                                  throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
OutboundEnablerProxyVO getOutboundEnablerProxy(java.lang.String
host,
                                               java.lang.Integer port)
                                                  throws
com.sybase.sup.admin.exception.SUPAdminException;
```

Parameters

- **outboundEnablerProxyId –** The ID of the outbound enabler proxy configuration you want to retrieve.
- **host –** The host name of the outbound enabler proxy configuration you want to retrieve.
- **port –** The port of the outbound enabler proxy configuration you want to retrieve.

Returns

If successful, retrieves a list of outbound enabler proxy configurations, or a specific outbound enabler proxy configuration. If unsuccessful, returns SUPAdminException.

Examples

- **Retrieval of a list of outbound enabler proxies –** retrieves a list of all outbound enabler proxy configurations.

```
@Test
public void getOutboundEnablerProxyVOByHostPort() throws
Exception {
    Iterator<OutboundEnablerProxyVO> iterator = supRelayServer
```

```
            .getOutboundEnablerProxies().iterator();
    if (!iterator.hasNext()) {
        System.out.println("No Outbound Enabler Proxy
configured.");
        return;
    }
    OutboundEnablerProxyVO proxy1 = iterator.next();
    OutboundEnablerProxyVO proxy2 =
supRelayServer.getOutboundEnablerProxy(
           proxy1.getHost(), proxy1.getPort());
    assertEquals(proxy1, proxy2);
}
```

- **Retrieval of an outbound enabler proxy with a given ID –** retrieves the outbound enabler proxy configuration with the given ID.

  ```
  OutboundEnablerVO rsvo_byId = suprs.getOutboundEnablerProxy(id);
  ```

- **Retrieval of an outbound enabler proxy with a given host and port –** retrieves the outbound enabler proxy configuration with the given host name and port.

  ```
  OutboundEnablerVO rsvo_hostPort =
  suprs.getOutboundEnablerProxy(rsvo.getHost(),
                  rsvo.getPort());
  ```

Create Outbound Enabler Proxy Configuration

Creates an outbound enabler proxy.

Syntax

```
void saveOutboundEnablerProxy(OutboundEnablerProxyVO
outboundEnablerProxy)
                               throws
com.sybase.sup.admin.exception.SUPAdminException;
```

Parameters

- **outboundEnablerProxy –** The outbound enabler proxy configuration to be created.

Returns

If successful, creates a new outbound enabler proxy configuration. If unsuccessful, returns SUPAdminException.

Examples

- **Creation of an outbound enabler proxy –** Creates an outbound server proxy configuration.

  ```
  Iterator<OutboundEnablerVO> iter = supServerConf
                  .getOutboundEnablers().iterator();
          if (!iter.hasNext())
              return;

          OutboundEnablerVO oe2 = new OutboundEnablerVO();
  ```

```
        OutboundEnablerVOBuilder bld = new
OutboundEnablerVOBuilder(oe2);
                bld.host ("proxy1").
                    port(4321)
                    credential ("proxy-user-1").
                       password("proxy-password-1")
                       back().
                    credential("proxy-user-2").
                       password("proxy-password-2")
                       back().

                suprs.saveOutboundEnablerProxy (bld.build()):
```

Update Outbound Enabler Proxy

Updates an existing outbound enabler proxy configuration with a given ID.

Syntax

```
void updateOutboundEnablerProxy(java.lang.Integer
outboundEnablerProxyId,
                                OutboundEnablerProxyVO
outboundEnablerProxy)
                                throws
com.sybase.sup.admin.exception.SUPAdminException;
```

Parameters

- **outboundEnablerProxyID –** The new ID of the outbound enabler proxy configuration.
- **outboundEnablerProxy –** The outbound enabler proxy configuration to be updated.

Returns

If successful, updates the outbound enabler proxy configuration. If unsuccessful, returns SUPAdminException.

Examples

- **Update of an outbound enabler proxy configuration –** updates the outbound enabler proxy with the given ID.

```
Iterator<OutboundEnablerVO> iter = supServerConf
                .getOutboundEnablers().iterator();
        if (!iter.hasNext())
            return;

        OutboundEnablerVO oe1 = iter.next();
        oe1.setCertificateFile("4321");

        supServerConf.updateOutboundEnabler(oe1.getID(), oe1);
```

Delete Outbound Enabler Proxy Configuration

Deletes one or more outbound enabler proxy configurations.

You can delete the following:

- a group of outbound enabler proxies based on a list
- a group of outbound enabler proxies based on a list of IDs
- a specific outbound enabler proxy based on an ID
- a specific outbound enabler proxy based on a host name and port number

Syntax

```
void
deleteOutboundEnablerProxies(java.util.List<OutboundEnablerProxyVO>
outboundEnablerProxies)
                                    throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
deleteOutboundEnablerProxies(java.util.Set<java.lang.Integer>
outboundEnablerProxyIds)
                                    throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
deleteOutboundEnablerProxy(java.lang.Integer
outboundEnablerProxyId)
                                   throws
com.sybase.sup.admin.exception.SUPAdminException;
```

```
void deleteOutboundEnablerProxy(java.lang.String host,
                                   java.lang.Integer port)
                                   throws
com.sybase.sup.admin.exception.SUPAdminException;
```

Parameters

- **outboundEnablerProxies –** The list of outbound enabler proxy configurations that you want to delete.
- **outboundEnablerProxyIds –** The list of outbound enabler proxy configurations IDs that you want to delete.
- **outboundEnablerProxyId –** The ID of the outbound enabler proxy configuration you want to delete.
- **host –** The host name of the outbound enabler proxy configuration you want to delete.
- **port –** The port of the outbound enabler proxy configurationn you want to delete.

Returns

If successful, deletes a list of outbound enabler proxies, or a specific outbound enabler proxy. If unsuccessful, returns SUPAdminException.

Examples

- **Deletion of a List of Outbound Enabler Proxies –** Deletes all outbound enabler proxy configurations in the list.

```
supRelayServer.deleteOutboundEnablerProxies(supRelayServer
         .getOutboundEnablerProxies());
```

- **Deletion of a List of Outbound Enabler Proxies by ID –** Deletes all outbound enabler proxy configurations with the given IDs.

```
Set<Integer> proxyIDsToDelete = new HashSet<Integer>();
proxyIDsToDelete.add(1);
proxyIDsToDelete.add(2);
supRelayServer.deleteOutboundEnablerProxies(proxyIDsToDelete);
```

- **Deletion of an Outbound Enabler Proxy with a Given ID –** Deletes the outbound enabler proxy configuration with the given ID.

```
Iterator<OutboundEnablerProxyVO> iter = supRelayServer
        .getOutboundEnablerProxies().iterator();
if (!iter.hasNext())
    return;
supRelayServer.deleteOutboundEnablerProxy(iter.next().getID());
```

- **Deletion of an Outbound Enabler Proxy with a Given Host and Port –** Deletes the outbound enabler proxy with the given host name and port.

```
supRelayServer.deleteOutboundEnablerProxy("myProxy.sybase.com",
80);
```

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;
```

Parameters

The deployment mode determines how the deployment process handles the objects in a deployment unit and package. Which value you choose depends on whether or not a package of the same name already exists on Unwired Server. Allowed values are:

- `UPDATE` – updates the target package with updated objects. After deployment, objects in the server's package with the same name as those being deployed are updated. By default, deploymentMode is UPDATE.
- `VERIFY` – do not deploy package. Only return errors, if any. Used to determine the results of the `UPDATE` deploy mode.

If the deployment mode is specified both in the descriptor file and the command-line, the command-line deploymentMode option override the deployment mode specified in the descriptor file.

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: Unwired Server Runtime > 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: Unwired Server Runtime > 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);
}
```

Set Default Security Configuration

Sets the default security configuration for a domain.

Syntax

```
void setDefaultSecurityConfiguration(String security) throws
SUPAdminException;
```

Returns

If successful, returns the default security configuration for a domain. If unsuccessful, returns `SUPAdminException`.

Examples

- **Set default security domain –** gets and sets the default security configuration for a domain.

```
supDomain.setDefaultSecurityConfiguration(defsec);
```

Retrieval of Default Security Configuration

Retrieves the default security configuration for a domain.

Syntax

```
String getDefaultSecurityConfiguration() throws SUPAdminException;
```

Returns

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

Examples

- **Retrieval of default security configuration –** retrieves the default security configuration for a domain:

```
supDomain.getDefaultSecurityConfiguration();
```

Update of Security Configurations

Updates security configurations in the domain. You must be assigned a Sybase Unwired Platform 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 Control Center for Sybase Unwired Platform > Deploy > MBO Packages > MBO Subscription Management > Managing 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 Control Center for Sybase Unwired Platform > Deploy > MBO Packages > MBO Subscription Management > Managing 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 Control Center for Sybase Unwired Platform > Administer > Security Configurations > Creating a Security Configuration > 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 Control Center for Sybase Unwired Platform > Get Started > About Sybase Control Center for Unwired Platform > 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`.

Export MBO Package

Exports a package from a domain. Only a Sybase Unwired Platform administrator or a Sybase Unwired Platform domain administrator can perform this.

Syntax

```
void exportPackage(String fileName, String name) 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:

```
supDomain.exportPackage(fileName, pkgName);
```

Import MBO Package

Imports a package to a domain. Only a Sybase Unwired Platform administrator or a Sybase Unwired Platform domain administrator can perform this.

Syntax

```
void importPackage(String fileName) 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(fileName);
```

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 *System Administration > EIS Connection Management > Data Source 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);
```

Export Hybrid App

Export a Hybrid App from the server. Only a Sybase Unwired Platform administrator or a Sybase Unwired Platform domain administrator can perform this.

Syntax

```
void exportMobileHybridApp(String fileName, MobileHybridAppIDVO
hybridAppID)
      throws SUPAdminException;
```

Parameters

- **fileName –** The name of the file which stores the exported Hybrid App.
- **hybridAppID –** The Hybrid App ID.

Returns

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

Examples

- **Export Hybrid App**

```
SUPMobileHybridApp hybridApp =
SUPObjectFactory.getSUPMobileHybridApp(clusterContext);
hybridApp.exportMobileHybridApp(fileName, hybridappID);
```

Import Hybrid App

Import an exported Hybrid App to the server. Only a Sybase Unwired Platform administrator or a Sybase Unwired Platform domain administrator can perform this.

Syntax

```
void importMobileHybridApp(String fileName)
      throws SUPAdminException;
```

Parameters

- **fileName –** The file name of the exported Hybrid App.

Returns

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

Examples

- **Import Hybrid App**

```
SUPMobileHybridApp hybridApp =
SUPObjectFactory.getSUPMobileHybridApp(clusterContext);
hybridApp.importMobileHybridApp(fileName);
```

Export Application

Export an application.

Syntax

```
void exportApplication(String fileName, String id)
      throws SUPAdminException;
```

Parameters

- **fileName –** The name of the file which stores the exported application.
- **id –** The application ID.

Returns

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

Examples

- **Export Application**

```
SUPApplication app =
SUPObjectFactory.getSUPApplication(clusterContext);
app.exportApplication(fileName, appID);
```

Import Application

Import an application archive.

Syntax

```
void importApplication(String fileName)
      throws SUPAdminException;
```

Parameters

- **fileName –** The file name of the exported application archive.

Returns

If successful, returns silently. If unsuccessful, throws SUPAdminException. If the import fails, the thrown exception will contain the proper error message.

Examples

- **Import Application**

```
SUPApplication app =
SUPObjectFactory.getSUPApplication(clusterContext);
app.importApplication(fileName);
```

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);
```

Retrieve Application Connection Setting

Retrieve the application connection setting for the specified field.

Syntax

```
Object getApplicationConnectionSettings(Integer numericId,
APPCONNECTION_SETTING_FIELD field)
            throws SUPAdminException;
```

Parameters

- **field –** the field for the setting.

Returns

If successful, returns the field setting. If unsuccessful, throws SUPAdminException. If the field is invalid, returns null.

Examples

- **Retrieve Notification Behavior –** Retrieves the NOTIFICATION_MODE setting:

```
public enum
com.sybase.sup.admin.enumeration.application.APPCONNECTION_SETTIN
G_FIELD {
...
/**
* Identifies the notification behaviour of the application
connection.
* <p>
* <br>
* Type is {@link APPCONNECTION_SETTING_FIELD_TYPE}.INTEGER 0: Only
Native
* Notifications 1: Only Online/Payload Push 2: Online/Payload Push
with
* Native Notifications
*/
@EnumAnn(alias = "Notification Mode")
```

```
NOTIFICATION_MODE(APPCONNECTION_SETTING_CATEGORY.APPLICATION,
APPCONNECTION_SETTING_FIELD_TYPE.INTEGER, 2907) {
},
...
}
```

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

Assign a Hybrid App to Application Connection Templates

Assign a Hybrid App to a list of application connection templates.

Syntax

```
void assignMobileHybridAppToTemplates(MobileHybridAppIDVO
hybridAppID, List<Integer> templateIDs)
      throws SUPAdminException;
```

Parameters

- **hybridAppID –** The Hybrid App ID.
- **templateIDs –** The list of application connection template IDs to assign to the Hybrid App.

Returns

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

Examples

- **Assign Hybrid App to Templates**

```
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setWID(1);
hybridAppID.setVersion(1);
List<Integer> templateIDs = new ArrayList<Integer>();
templateIDs.add(1);
hybridApp.assignMobileHybridAppToTemplates(hybridAppID,
templateIDs);
```

Unassign a Hybrid App from Application Connection Templates

Unassigns a Hybrid App from a list of application connection templates.

Syntax

```
void unassignMobileHybridAppFromTemplates(MobileHybridAppIDVO
hybridAppID, List<Integer> templateIDs)
          throws SUPAdminException;
```

Parameters

- **hybridAppID –** The Hybrid App ID.

- **templateIDs –** The list of application connection templates to unassign from the Hybrid App.

Returns

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

Examples

- **Unassign Hybrid App from Templates**

```
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setWID(1);
hybridAppID.setVersion(1);
List<Integer> templateIDs = new ArrayList<Integer>();
templateIDs.add(1);
hybridApp.unassignMobileHybridAppFromTemplates(hybridAppID,
templateIDs);
```

Retrieve List of Hybrid Apps Assigned to an Application Connection Template

Retrieve the Hybrid App assignment information for an application connection template. This can only be done by the Sybase Unwired Platform help desk or a Sybase Unwired Platform administrator.

Syntax

```
List<MobileHybridAppAssignmentVO>
getTemplateHybridAppAssignments(int templateID)
          throws SUPAdminException;
```

Parameters

- **templateID –** The application connection template ID.

Returns

If successful, returns a list of Hybrid Apps. If unsuccessful, throws SUPAdminException.

Examples

- **Get Template Hybrid App Assignments**

```
int templateID = 1;
List<MobileHybridAppAssignmentVO> list =
hybridApp.getTemplateHybridAppAssignments(templateID);
System. out .println(list.size());
```

Retrieve Status of Application Connection Templates for a Hybrid App

Retrieves a template status for a Hybrid App. This can only be done by the Sybase Unwired Platform helpdesk or a Sybase Unwired Platform administrator.

Syntax

```
 List<TemplateMobileHybridAppStatusVO>
getTemplateMobileHybridAppStatus(MobileHybridAppIDVO hybridAppID)
      throws SUPAdminException;
```

Parameters

- **hybridAppID –** The Hybrid App ID.

Returns

If successful, returns a list of application connection statuses. If unsuccessful, throws SUPAdminException.

Examples

- **Get Template Hybrid App Status**

```
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setWID(1);
hybridAppID.setVersion(1);
List<TemplateMobileHybridAppStatusVO> list =
hybridApp.getTemplateMobileHybridAppStatus(hybridAppID);
System.out.println(list.size());
```

Set Default Hybrid App for an Application Connection Template

Set the default Hybrid App for a list of application connection templates.

Syntax

```
 void setDefaultMobileHybridAppforTemplates(MobileHybridAppIDVO
hybridAppID, List<Integer> templateIDs)
          throws SUPAdminException;
```

Parameters

- **hybridAppID –** The Hybrid App ID.
- **templateIDs –** The list of application connection templates to set the default Hybrid App.

Returns

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

Examples

- **Set Default Hybrid App for Templates**

```
 MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setWID(1);
hybridAppID.setVersion(1);
List<Integer> templateIDs = new ArrayList<Integer>();
templateIDs.add(1);
hybridApp.setDefaultMobileHybridAppforTemplates(hybridAppID,
templateIDs);
```

Unset Default Hybrid App for an Application Connection Template

Remove the default Hybrid App for a list of application connection templates.

Syntax

```
void unsetDefaultMobileHybridAppforTemplates(List<Integer>
templateIDs)
      throws SUPAdminException
```

Parameters

- **templateIDs –** The list of application connection templates to remove the default Hybrid App.

Returns

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

Examples

- **Unset Default Hybrid App for Templates**

```
 List<Integer> templateIDs = new ArrayList<Integer>();
templateIDs.add(1);
hybridApp.unsetDefaultMobileHybridAppforTemplates(templateIDs);
```

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,
"mySecurity");
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);
```

Managing Application Customization Resource Bundles

Use the `SUPApplication` interface to manage customization resource bundles for OData SDK Android and iOS applications.

Retrieve an Application Customization Resource Bundle ID

Retrieves a customization resource bundle identifier from its binaries.

Syntax

```
String getCustomizationResourceBundleId(byte[]
customizationResourceBundleBytes)
  throws SUPAdminException;
```

Returns

If successful, returns the ID of the supplied customization resource bundle binaries. If unsuccessful, returns `SUPAdminException`.

Examples

- **Retrieval**

```
File file = new File("C:\\CustomizationResourceBundle.jar");
InputStream is = new FileInputStream(file);
byte[] bytes = new byte[is.available()];
is.read(bytes);
app.getCustomizationResourceBundleId(bytes);
```

Deploy an Application Customization Resource Bundle

Deploys a customization resource bundle to an application.

Syntax

```
void deployCustomizationResourceBundle(java.lang.String
applicationId,
  byte[] customizationResourceBundleBytes)
  throws SUPAdminException;
```

Returns

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

Examples

- **Deploy**

```
File file = new File("C:\\CustomizationResourceBundle.jar");
InputStream is = new FileInputStream(file);
```

```
byte[] bytes = new byte[is.available()];
is.read(bytes);
app.deployCustomizationResourceBundle("<application ID>", bytes);
```

Export an Application Customization Resource Bundle

Exports a customization resource bundle from an application to a JAR file.

Syntax

```
void exportCustomizationResourceBundle(java.io.File file,
  java.lang.String applicationId,
  java.lang.String customizationResourceBundleId)
  throws SUPAdminException;
```

Returns

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

Examples

- **Export**

```
File file = new File("C:\\ExportedCRB.jar");
app.exportCustomizationResourceBundle(file, "<application ID>",
"<customization resource bundle ID>");
```

Assign an Application Customization Resource Bundle

Assigns a customization resource bundle to all qualified application connections or application connection templates for the application ID.

Syntax

```
void assignCustomizationResourceBundle(java.lang.String
applicationId,
  java.lang.String customizationResourceBundleId,
  CUSTOMIZATION_RESOURCE_BUNDLE_REFERRAL referral)
  throws SUPAdminException;
```

Returns

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

Examples

- **Assign to all application connections with the defined application ID**

```
app.assignCustomizationResourceBundle("<application ID>",
"<customization resource bundle ID>",
CUSTOMIZATION_RESOURCE_BUNDLE_REFERRAL.APPCONNECTION);
```

- **Assign to all application connection templates with the defined application ID**

```
app.assignCustomizationResourceBundle("<application ID>",
"<customization resource bundle ID>",
CUSTOMIZATION_RESOURCE_BUNDLE_REFERRAL.APPCONNECTION_TEMPLATE);
```

Unassign an Application Customization Resource Bundle

Unassigns a customization resource bundle from all applicable application connections or application connection templates for the application ID.

Syntax

```
void unassignCustomizationResourceBundle(java.lang.String
applicationId,
  java.lang.String customizationResourceBundleId,
  CUSTOMIZATION_RESOURCE_BUNDLE_REFERRAL referral)
  throws SUPAdminException;
```

Returns

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

Examples

- **Unassign from all application connections with the defined application ID**

```
app.unassignCustomizationResourceBundle("<application ID>",
"<customization resource bundle ID>",
CUSTOMIZATION_RESOURCE_BUNDLE_REFERRAL.APPCONNECTION);
```

- **Unassign from all application connection templates with the defined application ID**

```
app.unassignCustomizationResourceBundle("<application ID>",
"<customization resource bundle ID>",
CUSTOMIZATION_RESOURCE_BUNDLE_REFERRAL.APPCONNECTION_TEMPLATE);
```

Delete an Application Customization Resource Bundle

Deletes a customization resource bundles from an application. You cannot delete a customization resource bundle if it is assigned to an application connection or application connection template; you must unassign it first.

Syntax

```
void deleteCustomizationResourceBundle(java.lang.String
applicationId,
  java.lang.String customizationResourceBundleId)
  throws SUPAdminException;
```

Returns

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

Examples

- **Delete**

```
app.deleteCustomizationResourceBundle("<application ID>",
"<customization resource bundle ID>");
```

Managing Application Push Configuration

Methods for managing application push configurations.

Create a push configuration

Creates a push configuration if there is not such a one, otherwise updates the existing one.

Syntax

```
public void
saveApplicationPushConfiguration(ApplicationPushConfigurationVO
      pushConfiguration) throws SUPAdminException;
```

Parameters

- **pushConfiguration –** Application push configuration to create or update.

Returns

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

Examples

- **Create an application push configuration**

```
app.saveApplicationPushConfiguration( "<ApplicationPushConfigurat
ionVO>" );
```

Delete a list of push configurations

Delete a list of push configurations from the specified application.

Syntax

```
 public void
deleteApplicationPushConfiguration(List<ApplicationPushConfiguratio
nVO> list)
      throws SUPAdminException;
```

Parameters

- **list –** Application push configuration objects.

Returns

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

Examples

- **Delete a list of push configurations**

```
app.deleteApplicationPushConfiguration("List<ApplicationPushConfi
gurationVO>");
```

Retrieve a list of push application configurations

Retrieves a list of push application configurations.

Syntax

```
 public List<ApplicationPushConfigurationVO>
getApplicationPushConfigurations(String applicationId,
APPLICATION_PUSH_CONFIGURATION_TYPE type)
            throws SUPAdminException;
```

Parameters

- **applicationId –** Application Identifier
- **type –** See APPLICATION_PUSH_CONFIGURATION_TYPE

Returns

If successful, returns a list of application push configurations. If unsuccessful, throws SUPAdminException.

Examples

- **Retrieve a list of push application configurations –** Retrieves a list of push application configurations for app.

```
 app.getApplicationPushConfigurations("<application ID>",
"<APPLICATION_PUSH_CONFIGURATION_TYPE>");
```

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: Unwired Server Runtime > 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: Unwired Server Runtime > 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 exportMessagingOperationStatistics(file, mc, null, null) throws
SUPAdminException;

void exportReplicationOperationStatistics(file, mc, null, null)
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, filter logs, and manage log appenders and buckets 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 appenders and buckets.

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 Appenders and Buckets

Sybase Unwired Platform server logs 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 appenders and buckets:

- 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: Unwired Server Runtime > 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();
```

Retrieval and Export of Trace Entries

Retrieves and exports trace entries to allow you to identify and resolve server-side issues while debugging a device application.

Syntax

```
PaginationResult<TraceEntryVO> getTraceEntries() throws
SUPAdminException;

void exportTraceEntries(exportFile, filter, sort) through
SUPAdminException
```

Returns

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

Examples

- **Retrieve and export trace entries**

```
TraceFilterVO filter = new TraceFilterVO();
Calendar c = Calendar.getInstance();
c.set(2011, 10, 1);

filter.setStartTime(c.getTime());
filter.setEndTime(new Date());
filter.setLevel(TRACE_LOG_LEVEL.DEBUG);
Collection<TRACE_LOG_MODULE> modules =
Arrays.asList(TRACE_LOG_MODULE.MO);
filter.setModules(modules);
TraceSortVO sort = null;
PaginationResult<TraceEntryVO> entries =
supServerLog.getTraceEntries(
filter, 0L, 100, sort);

System.out.println(entries.getTotalAvailableRecords());

File exportFile = new File("D:\\temp\\jmsBridge.zip");
supServerLog.exportTraceEntries(exportFile, filter, sort);
```

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.

Export of Log Entries

Exports the domain log entries to a file.

Syntax

```
File exportDomainLogEntry(file, Date StartTime, Date EndTime,
Integer length) throws SUPAdminException;
```

Returns

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

Examples

- **Export**

```
File file = new File("D:\\domainlog.txt");
domainLog.exportDomainLogEntry(file, Date StartTime, Date
EndTime, Integer length);
```

Configuring Unwired Servers

Administration of the Unwired Server configuration is provided through the `SUPServerConfiguration` interface.

The Unwired Server configuration consists of Java Virtual Machine (JVM) startup options and Outbound Enabler Proxy management, which are metadata-based configuration. All other components have been deprecated and are now performed on the cluster.

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: Unwired Server Runtime > 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: Unwired Server Runtime > 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

This method has been deprecated. 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

This method has been deprecated. 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 Messaging Sync Server Configuration

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated.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;
}
```

Start or Stop Outbound Enablers

Starts or stops Relay Server outbound enablers.

Syntax

```
void startOutboundEnablers (List<OutboundEnablerVO> outBoundEnabler)
throws SUPAdminException;
void stopOutboundEnablers (List<OutboundEnablerVO> outBoundEnabler)
throws SUPAdminException;
```

Parameters

- **outBoundEnabler –** The list of outbound enablers to start or stop.

Returns

If successful, starts or stops all the outbound enablers of the server. If unsuccessful, throws SUPAdminException.

Examples

- **Start all outbound enablers –** Start and stop all outbound enablers.

```
@Test
public void testStartAndStopOutboundEnabler() throws Exception {
    AgentContext agentContext = new
AgentContext("mySCC.sybase.com", 9999,
            "supAdmin", "s3pAdmin");
   ServerContext context = new ServerContext("mySUP.sybase.com",
2000,
            "supAdmin", "s3pAdmin", false, agentContext);
    SUPServerConfiguration serverConfig = SUPObjectFactory
            .getSUPServerConfiguration(context);
    // Get all Outbound Enablers of this Unwired Server node.
    List<OutboundEnablerVO> outBoundEnablers = serverConfig
            .getOutboundEnablers();
    // Start all the Outbound Enablers
    serverConfig.startOutboundEnablers(outBoundEnablers);
    // Stop all the Outbound Enablers
    serverConfig.stopOutboundEnablers(outBoundEnablers);
}
```

Retrieval of Relay Server Outbound Enablers

Retrieves one or more Relay Server outbound enabler configurations for the Unwired Server.

Syntax

```
List<OutboundEnablerVO> getOutboundEnablers() throws
SUPAdminException;
```

```
OutboundEnablerVO getOutboundEnabler() throws SUPAdminException;
```

Returns

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

Examples

- **Retrieval of all outbound enablers**

```
// get all Outbound Enablers of this server
List<OutboundEnablerVO> outboundEnablers = serverConfiguration
        .getOutboundEnablers();
```

- **Retrieval of a specific outbound enabler**

```
// get Outbound Enabler by specified ID.
OutboundEnablerVO outboundEnabler = serverConfiguration
        .getOutboundEnabler(3);
```

Creation or Update of Relay Server Outbound Enabler

Creates a new outbound enabler configuration if it does not exist; otherwise updates the existing one.

Syntax

```
void saveOutboundEnabler(OutboundEnablerVO outboundEnabler) throws
com.sybase.sup.admin.exception.SUPAdminException
```

Parameters

- **outboundEnabler –** the outbound enabler configuration to be created or updated.

Returns

If successful, creates or updates the specified outbound enabler configuration. If unsuccessful, returns `SUPAdminException` .

Examples

- **Update an outbound enabler configuration**

```
Iterator<OutboundEnablerVO> iter = supServerConf
        .getOutboundEnablers().iterator();
if (!iter.hasNext())
    return;

//update a outbound enabler configuration
OutboundEnablerVO oe1 = iter.next();
oe1.setUnwiredServerHost("<new host>");
supServerConf.saveOutboundEnabler(oe1);
```

- **Create an outbound enabler configuration**

```
//create a outbound enabler configuration
OutboundEnablerVO oe2 = new OutboundEnablerVO();
OutboundEnablerVOBuilder bld = new OutboundEnablerVOBuilder(oe2);
//copy setting from a existing one
bld.copy(oe1);
oe2.getServerNode().
    setName("<a different host name from the one copied>");
supServerConf.saveOutboundEnabler(oe2);
```

Update of Relay Server Outbound Enabler

Updates an existing Relay Server outbound enabler configuration.

Syntax

```
void updateOutboundEnabler(int outboundEnablerId, OutboundEnablerVO
outboundEnabler) throws
com.sybase.sup.admin.exception.SUPAdminException
```

Parameters

- **outboundEnablerId –** the ID of the outbound enabler configuration to be updated.
- **outboundEnabler –** the outbound enabler configuration containing the property values to be updated.

Returns

If successful, updates the specified outbound enabler configuration(s). If unsuccessful, returns `SUPAdminException`.

Examples

- **Update an outbound enabler configuration**

```
Iterator<OutboundEnablerVO> iter = supServerConf
        .getOutboundEnablers().iterator();
if (!iter.hasNext())
    return;

OutboundEnablerVO oe1 = iter.next();
oe1.setCertificateFile("4321");

supServerConf.updateOutboundEnabler(oe1.getID(), oe1);
```

Deletion of Relay Server Outbound Enabler

Deletes one or more Relay Server outbound enabler configurations.

You can delete the following:

- a group of outbound enabler configurations based on a list of IDs
- a specific outbound enabler configuration based on an ID

Syntax

```
void deleteOutboundEnablers(java.util.Set<java.lang.Integer>
outboundEnablerIds) throws
com.sybase.sup.admin.exception.SUPAdminException
```

```
void deleteOutboundEnabler(java.lang.Integer outboundEnablerId)
throws com.sybase.sup.admin.exception.SUPAdminException
```

Parameters

- **outboundEnablerIds –** The list of outbound enabler configuration IDs that you want to delete.
- **outboundEnablerId –** The ID of the outbound enabler configuration you want to delete.

Returns

If successful, deletes a list of outbound enabler configurations, or a specific outbound enabler configuration. If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion of a list of outbound enabler configurations by ID –** Deletes all outbound enabler configurations with the given IDs.

  ```
  serverConfiguration.deleteOutboundEnabler(1);
  Set<Integer> outboundEnablerIds=new HashSet<Integer>();
  outboundEnablerIds.add(2);
  outboundEnablerIds.add(3);
  serverConfiguration.deleteOutboundEnablers(outboundEnablerIds);
  ```

Addition of Relay Server Outbound Enabler Certificate Files

Adds a new certificate file to be used by Relay Server outbound enablers.

Syntax

```
void addOutboundEnablerCertificateFile(java.lang.String fileName,
byte[] certificateBlob, java.lang.Boolean overwrite) throws
com.sybase.sup.admin.exception.SUPAdminException
```

Parameters

- **fileName –** the file name to be used to save the certificate blob.
- **certificateBlob –** the certificate blob.

- **overwrite –** TRUE to overwrite existing file with the same name, FALSE to preserve the old one.

Returns

If successful, creates the specified certificate file. If unsuccessful, returns `SUPAdminException`.

Examples

- **Addition of an outbound enabler certificate file**

```
File file = new File("D:\\rsoe.cer");
byte[] blob = new byte[(int)file.length()];
new DataInputStream(new FileInputStream(file)).readFully(blob);
supServerConf.addOutboundEnablerCertificateFile(file.getName(),
blob, true);
```

Retrieval of Relay Server Outbound Enabler Certificate Files

Retrieves a specific Relay Server outbound enabler certificate file.

Syntax

```
List<java.lang.String> getOutboundEnablerCertificateFiles() throws
com.sybase.sup.admin.exception.SUPAdminException
```

Returns

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

Examples

- **Retrieval of outbound enabler certificate files**

```
List<String>
fileNames=serverConfiguration.getOutboundEnablerCertificateFiles(
);
```

Deletion of Relay Server Outbound Enabler Certificate Files

Deletes one or more Relay Server outbound enabler certificate files.

You can delete the following:

- a group of outbound enabler certificate files based on a list of file names.
- a specific outbound enabler certificate file based on a file name.

Syntax

```
void
deleteOutboundEnablerCertificateFiles(java.util.Set<java.lang.Strin
```

```
g> fileNames) throws
com.sybase.sup.admin.exception.SUPAdminException
```

```
void deleteOutboundEnablerCertificateFile(java.lang.String fileName)
throws com.sybase.sup.admin.exception.SUPAdminException
```

Parameters

- **fileNames –** The list of outbound enabler certificate file names that you want to delete.
- **fileName –** The file name of the outbound enabler certificate file you want to delete.

Returns

If successful, delete the specified file(s). If unsuccessful, returns `SUPAdminException`.

Examples

- **Deletion of a List of Outbound Enabler Certificate Files –** Deletes all outbound enabler certificates with the given file names.

```
serverConfiguration.deleteOutboundEnablerCertificateFile("rsoe.ce
r");
Set<String> certFileNames = new HashSet<String>();
certFileNames.add("rsoe2.cer");
certFileNames.add("rsoe3.cer");
serverConfiguration.deleteOutboundEnablerCertificateFiles(certFil
eNames);
```

Configuring Security Configurations

The Sybase Unwired Platform security configuration is a metadata-based configuration that includes several components.

- Authentication provider
- Authorization provider
- Audit provider
- Attribution provider (only if one has been developed with the CSI SDK)

Each of these components is a security provider, and is represented by `SecurityProviderVO`. The properties of `SecurityProviderVO` differentiate the components. See *Developer Guide: Unwired Server Runtime > 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: Unwired Server Runtime > 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();
```

Retrieval of Security Credentials

Syntax

```
difference(expr1,expr2)
```

Parameters

- **expr1 –** A character-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, or unichar type.
- **expr2 –** Another character-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, or unichar type.

Returns

If expr1 or expr2 are both not NULL, returns a value from 0 to 4 indicating the similarity between the two strings. The best match is 4.

If expr1 or expr2 is NULL, returns NULL.

Examples

- –

Usage

Standards

Security

Permissions

Auditing

Create logical role for security configuration

Creates logical role and sets the role mappings for this logical role.

Syntax

```
public void createLogicalRole(String logicalRole, ROLE_MAPPING_TYPE
type,
            List<String> physicalRoles) throws SUPAdminException;
```

Parameters

- **logicalRole –** The logical role name.
- **type –** One of the role mapping types: ROLE_MAPPING_TYPE.AUTO, ROLE_MAPPING_TYPE.MAPPED, or ROLE_MAPPING_TYPE.NONE.
- **physicalRoles –** The physical roles list if the type parameter is ROLE_MAPPING_TYPE.MAPPED, otherwise this parameter will be ignored.

Returns

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

Delete logical role for security configuration

Deletes logical role.

Syntax

```
public void deleteLogicalRole(String logicalRole) throws
SUPAdminException;
```

Parameters

- **logicalRole –** The logical role name to delete.

Returns

If successful, returns silently. If the logical role is already referred to by the application connection templates, throws SUPAdminException.

Retrieve role mappings for security configuration logical roles

Returns cluster level role mappings.

Syntax

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

Returns

If successful, returns the list of RoleMappingVO objects. If unsuccessful, returns SUPAdminException.

Update role mapping for security configuration logical role

Updates role mappings for security configuration.

Syntax

```
public void updateLogicalRole(String logicalRole, ROLE_MAPPING_TYPE
type,
            List<String> physicalRoles) throws SUPAdminException;
```

Parameters

- **logicalRole –** The logical role name.
- **type –** One of the role mapping types: ROLE_MAPPING_TYPE.AUTO, ROLE_MAPPING_TYPE.MAPPED, or ROLE_MAPPING_TYPE.NONE.
- **physicalRoles –** The physical roles list if the type parameter is ROLE_MAPPING_TYPE.MAPPED, otherwise this parameter will be ignored.

Returns

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

Managing Mobile Workflows

This interface has been deprecated. 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 `SUPMobileWorkflow` 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

This method has been deprecated. 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", "supPwd", 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This methods has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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

This method has been deprecated. 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");
```

Managing Hybrid Apps

Hybrid App packages, typically created through the Hybrid App Designer, allow a developer to design Hybrid App screens that can call on the create, update, and delete operations, as well as object queries, of a mobile business object.

You can manage Hybrid App packages through the `SUPMobileHybridApp` interface. Operations you can perform with this interface include:

- Starting administration of Hybrid App 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 Hybrid App device assignment
- Managing e-mail settings

Start Management of Hybrid App Packages

Starts the management of Hybrid App packages.

Syntax

```
public static SUPMobileHybridApp
getSUPMobileHybridApp(ClusterContext clusterContext) throws
SUPAdminException;
```

Returns

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

Examples

- **Start Hybrid App package management**

```
...
private SUPMobileHybridApp mobileHybridApp;
...
ServerContext serverContext = new ServerContext("wangf-dell",
2000, "supAdmin", "supPwd", false);
clusterContext = serverContext.getClusterContext("my cluster");
mobileHybridApp =
SUPObjectFactory.getSUPMobileHybridApp(clusterContext);
```

Usage

To manage Unwired Server Hybrid App packages, you must create an instance of `ServerContext` with the correct information, and pass it to `SUPObjectFactory.getSUPMobileHybridApp()`. When an instance of `SUPMobileHybridApp` is returned, you can call its method as a typical Java method call.

Hybrid App Package Retrieval

Retrieves a list of Hybrid App packages.

Syntax

```
List<MobileHybridAppVO> getMobileHybridAppList() throws
SUPAdminException;
```

Returns

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

Examples

- **Hybrid App package retrieval**

```
List<MobileHybridAppVO>
hybridApps=mobileHybridApp.getMobileHybridAppList();
```

Installation of a Hybrid App Package

Installs a Hybrid App package.

Syntax

```
MobileHybridAppIDVO installMobileHybridApp(byte[]
zippedWorkflowPackage) throws SUPAdminException;
```

Returns

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

Examples

- **Hybrid App package installation –** This code fragment installs a Hybrid App package named `ActivitiesPackage.zip`, and returns the package name once it is successfully installed:

```
// Install workflow
    byte[] hybridAppBytes= getHybridAppBytes();
    MobileHybridAppIDVO hybridAppID = mobileHybridApp
                    .installMobileHybridApp(hybridAppBytes);

   private byte[] getHybridAppBytes() 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 Hybrid App Package

Deletes the specified Hybrid App package.

Syntax

```
void deleteMobileHybridApp(MobileHybridAppIDVO hybridAppID) throws
SUPAdminException;
```

Returns

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

Examples

- **Hybrid App package deletion –** This code fragment deletes a Hybrid App package with the specified Hybrid App ID:

```
mobileHybridApp.deleteMobileHybridApp(hybridAppID);
```

Retrieval of Hybrid App Matching Rules

Retrieves matching rules for the specified Hybrid App 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

```
MobileHybridAppMatchingRulesVO
getMobileHybridAppMatchingRule(MobileHybridAppIDVO hybridAppID)
throws SUPAdminException;
```

Returns

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

Examples

- **Mobile Hybrid App matching rules**

```
// Get HybridApp Matching rule
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setVersion(1);
hybridAppID.setWID(6);
MobileHybridAppMatchingRulesVO vo =
mobileHybridApp.getMobileHybridAppMatchingRule(hybridAppID);
```

Retrieval of Hybrid App Context Variables

Retrieves context variables for the specified Hybrid App 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<MobileHybridAppContextVariableVO>
getMobileHybridAppContextVariables(MobileHybridAppIDVO hybridAppID)
throws SUPAdminException;
```

Returns

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

Examples

- **Hybrid App context variables –** This code fragment retrieves context variables for the specified Hybrid App package:

```
// Get HybridApp context variables
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setVersion(1);
hybridAppID.setWID(6);
List<MobileHybridAppContextVariableVO> list = mobileHybridApp
        .getMobileHybridAppContextVariables(hybridAppID);
```

Retrieval of Hybrid App Client Variables

Retrieves the client variables for the specified Hybrid App package.

Client variables are defined for Hybrid Apps by Hybrid App developers.

Syntax

```
List<MobileHybridAppClientVariableVO>
getMobileHybridAppClientVariables(
MobileHybridAppIDVO hybridAppID) throws SUPAdminException;
```

Parameters

- **hybridAppID –** The ID of the Hybrid App.

Returns

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

Examples

- **Hybrid App client variables –** This code fragment retrieves context variables for the specified Hybrid App package:

```
MobileHybridAppIDVO hybridAppId = new MobileHybridAppIDVO();
hybridAppId.setWID(1);
hybridAppId.setVersion(1);
List<MobileHybridAppClientVariableVO> clientVariables = workflow
       .getMobileHybridAppClientVariables(hybridAppId);
```

Usage

To retrieve Hybrid App client variables, you must be assigned an SUP Administrator or SUP Helpdesk role.

Retrieval of a Hybrid App Error List

Retrieves an error list for the specified Hybrid App package for the specified time period, and paginates the results.

Syntax

```
PaginationResult<MobileHybridAppErrorVO>
getMobileHybridAppErrorList(int startIndex, int maxRecordsToReturn,
MobileHybridAppIDVO 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

- **Hybrid App error list –** retrieves an error list for the Hybrid App package starting from the date September 30, 2009:

```
// Get HybridApp error list
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setVersion(1);
hybridAppID.setWID(6);
Calendar startDate = Calendar.getInstance();
startDate.set(2009, 9, 30);
Calendar endDate = Calendar.getInstance();
PaginationResult<MobileHybridAppErrorVO> result = mobileHybridApp
                .getMobileHybridAppErrorList(0, 1, hybridAppID,
"TEST4", startDate,
                         endDate, null, true);
```

Retrieval and Management of Hybrid App Queue Items

Retrieves a list of queue items for the specified Hybrid App package, and deletes the specified queue items.

Syntax

```
PaginationResult<MobileHybridAppQueueItemVO>
getMobileHybridAppQueueItems(int startIndex, int maxRecordsToReturn,
MobileHybridAppIDVO id, List<Integer> deviceIDs, List<String>
userNames, String orderByField, boolean ascending) throws
SUPAdminException;
   void deleteMobileHybridAppQueueItem(Integer queueItemID, Boolean
forTransformQueue) throws SUPAdminException;
```

Returns

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

Examples

- **Hybrid App queue items**

```
// Get HybridApp queue items
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setVersion(1);
hybridAppID.setWID(1);
List<Integer> deviceIds = new ArrayList<Integer>();
deviceIds.add(4);
PaginationResult<MobileHybridAppQueueItemVO> list =
mobileHybridApp
        .getMobileHybridAppQueueItems(0, 2, hybridAppID, null,
null, null, false);

//Delete HybridApp queue items.
mobileHybridApp.deleteMobileHybridAppQueueItem(1, true);
```

Update of Hybrid App Properties

Updates the properties for the specified Hybrid App package.

Syntax

```
void updateMobileHybridAppDisplayName(MobileHybridAppIDVO
hybridAppID, String displayName) throws SUPAdminException;
void updateMobileHybridAppIconIndex(MobileHybridAppIDVO hybridAppID,
int iconIndex) throws SUPAdminException;
```

Returns

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

Examples

- **Hybrid App properties –** updates the display name and icon index for the specified Hybrid App package:

```
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setVersion(1);
hybridAppID.setWID(6);

// Update workflow display name
mobileHybridApp.updateMobileHybridAppDisplayName(hybridAppID, "My
Hybrid App");

// Update workflow icon index
mobileHybridApp.updateMobileHybridAppIconIndex(hybridAppID, 100);
```

Update of Hybrid App Matching Rules

Updates a matching rule for the specified Hybrid App package.

Syntax

```
void updateMobileHybridAppMatchingRule(MobileHybridAppIDVO
hybridAppID, MobileHybridAppMatchingRulesVO matchRule) throws
SUPAdminException;
```

Returns

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

Examples

- **Hybrid App matching rules**

```
// Update HybridApp matching rule
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setVersion(1);
hybridAppID.setWID(6);
MobileHybridAppMatchingRulesVO matchRule = mobileHybridApp
        .getMobileHybridAppMatchingRule(hybridAppID);
matchRule.setBODYExpressionType(MobileHybridAppMatchingRulesVO.EX
PRESSION_TYPE_REGULAREXPRESSION);
matchRule.setBODYExpression(".*wang.*");
mobileHybridApp.updateMobileHybridAppMatchingRule(hybridAppID,
matchRule);
```

Update of Hybrid App Context Variables

Updates context variables for the specified Hybrid App package.

Syntax

```
void updateMobileHybridAppContextVariables(MobileHybridAppIDVO
hybridAppID,List<MobileHybridAppContextVariableVO>
contextVariables) throws SUPAdminException;
```

Returns

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

Examples

- **Hybrid App context variables –** updates context variables for an existing Hybrid App package with workflow ID 2:

```
// Update HybridApp context variables
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
// ID 2 version 1 is a existing HybridApp on the server
```

```
hybridAppID.setVersion(1);
hybridAppID.setWID(2);
List<MobileHybridAppContextVariableVO> contextVariables =
mobileHybridApp
        .getMobileHybridAppContextVariables(hybridAppID);
contextVariables.get(0).setValue("string value updated");
mobileHybridApp.updateMobileHybridAppContextVariables(hybridAppID
,contextVariables);
```

Usage

For Hybrid App packages that do not support certificate-based authentication, use the following context variables to specify credentials:

- SupUser
- SupPassword

For Hybrid App 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.

Update of Hybrid App Client Variables

Updates client variables for the specified Hybrid App by first deleting the current client variables and then setting the new client variables. It also updates the version number of the client variables.

Client variables are defined for Hybrid Apps by Hybrid App developers. Administrators can modify client variable values or add new client variables for the production environment.

Syntax

```
int setMobileHybridAppClientVariables(MobileHybridAppIDVO
hybridAppID, List<MobileHybridAppClientVariableVO> clientVariables,
Boolean bSendNotificationToDeployedClient) throws
SUPAdminException;
```

Parameters

- **hybridAppID –** The ID of the Hybrid App.
- **clientVariable –** The new client variable.
- **bSendNotificationToDeployedClient –** Pushes new client variable to each previously deployed client.

Returns
The version number of the client variables.

Examples

- **Create new Hybrid App client variable –** This code fragment create a new client variable for the specified Hybrid App but does not push it to deployed clients.

```
MobileHybridAppIDVO hybridAppId = new MobileHybridAppIDVO();
hybridAppId.setWID(1);
hybridAppId.setVersion(1);
List<MobileHybridAppClientVariableVO> clientVariables = new
ArrayList<MobileHybridAppClientVariableVO>();
MobileHybridAppClientVariableVO vo = new
MobileHybridAppClientVariableVO();
vo.setName("Name");
vo.setValue("value");
clientVariables.add(vo);
workflow.setMobileHybridAppClientVariables(hybridAppId,
       clientVariables, true);
```

Usage
To update Hybrid App client variables, you must be assigned an SUP administrator role.

Retrieval of Hybrid App Device Status
Retrieves Hybrid App status for a device from the value object `DeviceMobileHybridAppStatusVO`.

Syntax

```
List<MobileHybridAppAssignmentVO>
getDeviceHybridAppAssignments(Integer deviceLogicalID) throws
SUPAdminException;
```

Returns

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

Examples

- **Hybrid App device assignments**

```
// get all HybridApp that assign to the device. Where 3 is a
// existing device ID.
List<MobileHybridAppAssignmentVO> assignments = mobileHybridApp
        .getDeviceHybridAppAssignments(3);
```

Assignment of a Hybrid App 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 Hybrid App 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 Hybrid App 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 for Hybrid Apps

Updates or retrieves the e-mail settings for a Hybrid App 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

- **Hybrid App e-mail settings**

```
String configXmlString = readEmailConfig();

// Test Email Multicast connection
Boolean test = mobileHybridApp.testEmailConnection(config);

// Config Email Multicast
mobileHybridApp.configureEmail(config);

// Enable Email Multicast
mobileHybridApp.enableEmail(true);

// Get Email Multicast configuration
String config = mobileHybridApp.getEmailConfiguration();

// Check if Email Multicast enabled
boolean enable = mobileHybridApp.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 Hybrid App Queue

Unblocks the Hybrid App queue for the selected Hybrid Apps and devices.

Syntax

```
void unblockHybridAppQueueForDevices(MobileHybridAppIDVO
hybridAppID, List<Integer> deviceIDs, Boolean forTransformQueue)
throws SUPAdminException;
```

Returns

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

Examples

- **Hybrid App queue**

```
// prepare mobile HybridApp ID
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setVersion(1);
hybridAppID.setWID(2);
// prepare device ids
List<Integer> deviceIDs = new ArrayList<Integer>();
deviceIDs.add(1);
deviceIDs.add(2);
// Unblock mobile HybridApp queue for devices
mobileHybridApp.unblockHybridAppQueueForDevices(hybridAppID,
deviceIDs, true);
```

Replace Hybrid App Certificate

Replaces the certificate for a Hybrid App package.

Syntax

```
void replaceMobileHybridAppCertificate(MobileHybridAppIDVO
hybridAppID, byte[] certificateBlob, String certificatePassword)
throws SUPAdminException;
```

Returns

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

Examples

- **Replace certificate**

```
InputStream is = getClass().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();
MobileHybridAppIDVO hybridAppID = new MobileHybridAppIDVO();
hybridAppID.setWID(4);
hybridAppID.setVersion(1);

mobileHybridApp.replaceMobileHybridAppCertificate(hybridAppID,
        baos.toByteArray(), "password");
```

Management Client Application Shutdown

Releases resources currently held by the API. This method only needs to be called on the termination of the management 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 41
- *Metadata* on page 42

Security Configuration

The security configuration for Sybase Unwired Platform consists of the several types of security providers.

- Audit provider
- Authentication provider
- Authorization provider
- Attribution 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 supported value for destination is `com.sybase.security.core.FileAuditDestination`. Optionally, you can develop a custom provider and configure it as the audit destination, formatter, and filter. See *CSI Audit Generation and Configuration*.
- 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 class contains the following configurable properties:

Marking an audit destination required or requisite means all operations being audited (authorization, authentication, role check, and so on) will fail if the event cannot successfully be logged to that audit destination.

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. implementationClass

Datatype	String
Default	com.sybase.security.core.FileAuditDestination
Required?	Yes
Requires server restart?	No

Read-only?	No

Table 19. providerType

Datatype	String
Default	AuditDestination
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 20. auditFile

Datatype	String
Default	${djc.home}/logs/{security configuration name}-audit.log, where {security configuration name} is the name of the security configuration in which the audit destination is configured.
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 21. compressionThreshold

Datatype	int
Required?	No
Requires server restart?	No
Read-only?	No

Table 22. deleteThreshold

Datatype	int
Required?	No
Requires server restart?	No
Read-only?	No

Table 23. encoding

Datatype	String

Default	utf-8
Required?	No
Requires server restart?	No
Read-only?	No

Table 24. errorThreshold

Datatype	int
Required?	No
Requires server restart?	No
Read-only?	No

Table 25. logSize

Datatype	long
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.core.DefaultAuditFilter

The com.sybase.security.core.DefaultAuditFilter class contains the following configurable properties:

Table 26. implementationClass

Datatype	String
Default	com.sybase.security.core.DefaultAuditFilter
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 27. providerType

Datatype	String
Default	AuditFilter
Required?	Yes

Requires server restart?	No
Read-only?	Yes

Table 28. caseSensitiveFiltering

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 29. 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 class contains the following configurable properties:

Table 30. implementationClass

Datatype	String
Default	com.sybase.security.core.XmlAuditFormatter
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 31. 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.ClientValuePropagatingLoginModule
- com.sybase.security.radius.RadiusLoginModule

com.sybase.security.core.NoSecLoginModule

The com.sybase.security.core.NoSecLoginModule package includes the following configurable properties:

Table 32. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 33. controlFlag

Datatype	String (enumerated)

Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 34. identity

Datatype	String
Default	nosec_identity
Required?	No
Requires server restart?	No
Read-only?	No

Table 35. implementationClass

Datatype	String
Default	com.sybase.security.core.NoSecLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 36. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 37. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 38. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 39. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 40. 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 41. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 42. implementationClass

Datatype	String
Default	com.sybase.security.core.CertificateValidationLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 43. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 44. validatedCertificateIsIdentity

Datatype	boolean

Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 45. enableRevocationChecking

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 46. trustedCertStore

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 47. trustedCertStorePassword

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 48. trustedCertStoreProvider

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 49. trustedCertStoreType

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 50. 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 51. AuthenticationFilter

Datatype	String.
Required?	No
Requires server restart?	No
Read-only?	No

Table 52. AuthenticationMethod

Datatype	String
Default	simple
Required?	No
Requires server restart?	No
Read-only?	No

Table 53. AuthenticationScope

Datatype	String (enumerated)

Alllowable values	• onelevel • subtree
Default	onelevel
Required?	No
Requires server restart?	No
Read-only?	No

Table 54. AuthenticationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 55. BindDN

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 56. BindPassword

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No
Encrypted?	Yes

Table 57. CertificateAuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No

Read-only?	No

Table 58. ConnectTimeout

Datatype	Int
Default	0
Required?	No
Requires server restart?	No
Read-only?	No

Table 59. DefaultSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 60. DigestMD5AuthenticationFormat

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 61. EnableLDAPConnectionTrace

Datatype	Boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 62. InitialContextFactory

Datatype	String
Default	com.sun.jndi.ldap.LdapCtxFactory

Required?	No
Requires server restart?	No
Read-only?	No

Table 63. LDAPPoolMaxActive

Datatype	Int
Default	8
Required?	No
Requires server restart?	No
Read-only?	No

Table 64. ProviderURL

Datatype	String
Default	ldap://localhost:389
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 65. ReadTimeout

Datatype	Int
Required?	No
Requires server restart?	No
Read-only?	No

Table 66. Referral

Datatype	String (enumerated)
Allowable values	• ignore • follow • throw
Default	ignore
Required?	No

Requires server restart?	No
Read-only?	No

Table 67. RoleFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 68. RoleMemberAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 69. RoleNameAttributes

Datatype	String
Default	cn
Required?	No
Requires server restart?	No
Read-only?	No

Table 70. RoleScope

Datatype	String (enumerated)
Allowable values	• onelevel • subtree
Required?	No
Requires server restart?	No
Read-only?	No

Table 71. RoleSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 72. SecurityProtocol

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 73. SelfRegistrationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 74. SerializationKey

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 75. ServerType

Datatype	String (enumerated)
Allowable values	• sunone5 • msad2k • nsds4 • openldap
Required?	No

Requires server restart?	No
Read-only?	No

Table 76. SkipRoleLookup

Datatype	Boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 77. UnmappedAttributePrefix

Datatype	String
Default	LDAP
Required?	No
Requires server restart?	No
Read-only?	No

Table 78. UseUserAccountControlAttribute

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 79. UserFreeformRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 80. UserRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 81. certificateAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 82. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 83. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 84. enableCertificateAuthentication

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 85. implementationClass

Datatype	String
Default	com.sybase.security.ldap.LDAPLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 86. ldapAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 87. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 88. storePass

Datatype	boolean
Default	FALSE
Required?	No

Requires server restart?	No
Read-only?	No

Table 89. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 90. 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 91. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 92. controlFlag

Datatype	String (enumerated)

Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 93. defaultAuthenticationServer

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 94. defaultDomain

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 95. extractDomainFromUsername

Datatype	boolean
Default	TRUE
Required?	No
Requires server restart?	No
Read-only?	No

Table 96. failAuthenticationIfNoRoles

Datatype	boolean
Default	FALSE

Required?	No
Requires server restart?	No
Read-only?	No

Table 97. implementationClass

Datatype	String
Default	com.sybase.security.os.NTProxyLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 98. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 99. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 100. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No

Read-only?	No

Table 101. 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 102. DisableServerCertificateValidation

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 103. SapSSOTokenPersistenceDataStore

Datatype	String
Default	jdbc/default
Required?	No
Requires server restart?	No
Read-only?	Yes

Table 104. SapServerCertificate

Datatype	String
Required?	No

Requires server restart?	No
Read-only?	No

Table 105. SapServerCertificatePassword

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 106. SapServerURL

Datatype	String
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 107. TokenExpirationInterval

Datatype	long
Default	120
Required?	No
Requires server restart?	No
Read-only?	No

Table 108. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 109. controlFlag

Datatype	String (enumerated)

Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 110. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.sap.SAPSSOTokenLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 111. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 112. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 113. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 114. 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 115. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 116. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional

Required?	Yes
Requires server restart?	No
Read-only?	No

Table 117. enableRevocationChecking

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 118. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.core.CertificateAuthenticationLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 119. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 120. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No

Read-only?	No

Table 121. trustedCertStore

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 122. trustedCertStorePassword

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 123. trustedCertStoreProvider

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 124. trustedCertStoreType

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 125. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No

Read-only?	No

Table 126. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 127. 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 128. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 129. controlFlag

Datatype	String (enumerated)

Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 130. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.core.PreConfiguredUserLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 131. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 132. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 133. trustedCertStore

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 134. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 135. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 136. username

Datatype	String. Cannot contain , = : ' " * ? &.
Default	supAdmin
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 137. Password

Datatype	String
Default	“”
Required?	Yes

Requires server restart?	No
Read-only?	No

Table 138. roles

Datatype	String
Default	SUP Administrator
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.http.HttpAuthenticationLoginModule

The com.sybase.security.http.HttpAuthenticationLoginModule package contains the following configurable properties:

Table 139. implementationClass

Datatype	String (enumerated)
Default	com.sybase.security.http.HttpAuthenticationLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 140. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Table 141. controlFlag

Datatype	String (enumerated)

Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 142. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 143. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 144. storePass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 145. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 146. URL

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 147. DisableServerCertificateValidation

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 148. RolesHTTPHeader

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 149. SSOCookieNames

Datatype	String
Default	None

Required?	No
Requires server restart?	No
Read-only?	No

Table 150. SuccessfulConnectionStatusCode

Datatype	int
Default	200
Required?	No
Requires server restart?	No
Read-only?	No

Table 151. CredentialName

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 152. SendPasswordAsCookie

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 153. ClientHttpValuesToSend

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No

Read-only?	No

Table 154. SendClientHttpValueAs

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 155. UsernameHttpHeader

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 156. TokenExpirationTimeHttpHeader

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 157. RegexForUsernameMatch

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 158. TryBasicAuthIfTokenAuthFails

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 159. HttpConnectionTimeoutInterval

Datatype	int
Default	60000
Required?	No
Requires server restart?	No
Read-only?	No

Table 160. TokenExpirationalInterval

Datatype	int
Default	0
Required?	No
Requires server restart?	No
Read-only?	No

com.sybase.security.radius.RadiusLoginModule

The com.sybase.security.radius.RadiusLoginModule package contains the following configurable properties:

Table 161. implementationClass

Datatype	String
Default	com.sybase.security.radius.RadiusLoginModule
Required?	Yes
Requires server restart?	No
Read-only?	No

Encrypted?	No

Table 162. providerType

Datatype	String
Default	LoginModule
Required?	Yes
Requires server restart?	No
Read-only?	Yes
Encrypted?	No

Table 163. controlFlag

Datatype	String (enumerated)
Allowable values	• optional • sufficient • required • requisite
Default	optional
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 164. AuthenticationMethod

Datatype	String
Allowable values	• PAP • CHAP
Default	PAP
Required?	No
Requires server restart?	No
Read-only?	No
Encrypted?	No

Table 165. SharedSecret

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No
Encrypted?	Yes

Table 166. RadiusServerHostName

Datatype	String
Default	None
Required?	Yes
Requires server restart?	No
Read-only?	No
Encrypted?	No

Table 167. RadiusServerAuthPort

Datatype	int
Default	1812
Required?	Yes
Requires server restart?	No
Read-only?	No
Encrypted?	No

Table 168. RadiusServerAuthPort

Datatype	int
Default	3
Required?	No
Requires server restart?	No
Read-only?	No

Encrypted?	No

Table 169. caseSensitiveMatching

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No
Encrypted?	No

Table 170. useFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No
Encrypted?	No

Table 171. tryFirstPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No
Encrypted?	No

Table 172. storePass

Datatype	boolean
Default	FALSE
Required?	No

Requires server restart?	No
Read-only?	No
Encrypted?	No

Table 173. clearPass

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No
Encrypted?	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 174. implementationClass

Datatype	String
Default	com.sybase.security.core.NoSecAuthorizer
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 175. 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 176. AuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 177. AuthenticationMethod

Datatype	String
Default	simple
Required?	No
Requires server restart?	No
Read-only?	No

Table 178. AuthenticationScope

Datatype	String (enumerated)
Alllowable values	• onelevel • subtree
Default	onelevel
Required?	No
Requires server restart?	No
Read-only?	No

Table 179. AuthenticationSearchBase

Datatype	String
Required?	No

Requires server restart?	No
Read-only?	No

Table 180. BindDN

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 181. BindPassword

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 182. CertificateAuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 183. ConnectTimeout

Datatype	Int
Default	0
Required?	No
Requires server restart?	No
Read-only?	No

Table 184. DefaultSearchBase

Datatype	String
Required?	Yes (if RoleSearchBase is not provided)

Requires server restart?	No
Read-only?	No

Table 185. DigestMD5AuthenticationFormat

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 186. EnableLDAPConnectionTrace

Datatype	Boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 187. InitialContextFactory

Datatype	String
Default	com.sun.jndi.ldap.LdapCtxFactory
Required?	No
Requires server restart?	No
Read-only?	No

Table 188. LDAPPoolMaxActive

Datatype	Int
Default	8
Required?	No
Requires server restart?	No
Read-only?	No

Table 189. ProviderURL

Datatype	String
Default	ldap://localhost:389
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 190. ReadTimeout

Datatype	Int
Required?	No
Requires server restart?	No
Read-only?	No

Table 191. Referral

Datatype	String (enumerated)
Allowable values	• ignore • follow • throw
Default	ignore
Required?	No
Requires server restart?	No
Read-only?	No

Table 192. RoleFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 193. RoleMemberAttributes

Datatype	String

Required?	No
Requires server restart?	No
Read-only?	No

Table 194. RoleNameAttributes

Datatype	String
Default	cn
Required?	No
Requires server restart?	No
Read-only?	No

Table 195. RoleScope

Datatype	String (enumerated)
Allowable values	• onelevel • subtree
Required?	No
Requires server restart?	No
Read-only?	No

Table 196. RoleSearchBase

Datatype	String
Required?	Yes (if DefaultSearchBase is not provided)
Requires server restart?	No
Read-only?	No

Table 197. SecurityProtocol

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 198. SelfRegistrationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 199. ServerType

Datatype	String (enumerated)
Allowable values	• sunone5 • msad2k • nsds4 • openldap
Required?	No
Requires server restart?	No
Read-only?	No

Table 200. SkipRoleLookup

Datatype	Boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 201. UnmappedAttributePrefix

Datatype	String
Default	LDAP
Required?	No
Requires server restart?	No
Read-only?	No

Table 202. UseUserAccountControlAttribute

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 203. UserFreeformRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 204. UserRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 205. certificateAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 206. enableCertificateAuthentication

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 207. implementationClass

Datatype	String
Default	com.sybase.security.ldap.LDAPAuthorizer
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 208. ldapAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 209. providerType

Datatype	String
Default	Authorizer
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Attribution Provider

Supported attributions.

- com.sybase.security.core.NoSecAttributer
- com.sybase.security.ldap.LDAPAttributer

com.sybase.security.core.NoSecAttributer

The com.sybase.security.core.NoSecAttributer package contains the following configurable properties:

Table 210. implementationClass

Datatype	String
Default	com.sybase.security.core.NoSecAttributer

Required?	Yes
Requires server restart?	No
Read-only?	No

Table 211. providerType

Datatype	String
Default	Attributer
Required?	Yes
Requires server restart?	No
Read-only?	Yes

com.sybase.security.ldap.LDAPAttributer

The com.sybase.security.ldap.LDAPAttributer package contains the following configurable properties:

Table 212. AuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 213. AuthenticationMethod

Datatype	String
Default	simple
Required?	No
Requires server restart?	No
Read-only?	No

Table 214. AuthenticationScope

Datatype	String (enumerated)
Alllowable values	• onelevel • subtree

Default	onelevel
Required?	No
Requires server restart?	No
Read-only?	No

Table 215. AuthenticationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 216. BindDN

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 217. BindPassword

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 218. CertificateAuthenticationFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 219. ConnectTimeout

Datatype	Int

Default	0
Required?	No
Requires server restart?	No
Read-only?	No

Table 220. DefaultSearchBase

Datatype	String
Required?	Yes (if RoleSearchBase is not provided)
Requires server restart?	No
Read-only?	No

Table 221. DigestMD5AuthenticationFormat

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 222. EnableLDAPConnectionTrace

Datatype	Boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 223. InitialContextFactory

Datatype	String
Default	com.sun.jndi.ldap.LdapCtxFactory
Required?	No
Requires server restart?	No
Read-only?	No

Table 224. LDAPPoolMaxActive

Datatype	Int
Default	8
Required?	No
Requires server restart?	No
Read-only?	No

Table 225. ProviderURL

Datatype	String
Default	ldap://localhost:389
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 226. ReadTimeout

Datatype	Int
Required?	No
Requires server restart?	No
Read-only?	No

Table 227. Referral

Datatype	String (enumerated)
Allowable values	• ignore • follow • throw
Default	ignore
Required?	No
Requires server restart?	No
Read-only?	No

Table 228. RoleFilter

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 229. RoleMemberAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 230. RoleNameAttributes

Datatype	String
Default	cn
Required?	No
Requires server restart?	No
Read-only?	No

Table 231. RoleScope

Datatype	String (enumerated)
Allowable values	• onelevel • subtree
Required?	No
Requires server restart?	No
Read-only?	No

Table 232. RoleSearchBase

Datatype	String
Required?	Yes (if DefaultSearchBase is not provided)

Requires server restart?	No
Read-only?	No

Table 233. SecurityProtocol

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 234. SelfRegistrationSearchBase

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 235. ServerType

Datatype	String (enumerated)
Allowable values	• sunone5 • msad2k • nsds4 • openldap
Required?	No
Requires server restart?	No
Read-only?	No

Table 236. SkipRoleLookup

Datatype	Boolean
Default	False
Required?	No
Requires server restart?	No
Read-only?	No

Table 237. UnmappedAttributePrefix

Datatype	String
Default	LDAP
Required?	No
Requires server restart?	No
Read-only?	No

Table 238. UseUserAccountControlAttribute

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 239. UserFreeformRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 240. UserRoleMembershipAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 241. certificateAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 242. enableCertificateAuthentication

Datatype	boolean
Default	FALSE
Required?	No
Requires server restart?	No
Read-only?	No

Table 243. implementationClass

Datatype	String
Default	com.sybase.security.ldap.LDAPAttributer
Required?	Yes
Requires server restart?	No
Read-only?	No

Table 244. ldapAttributes

Datatype	String
Required?	No
Requires server restart?	No
Read-only?	No

Table 245. providerType

Datatype	String
Default	Attributer
Required?	Yes
Requires server restart?	No
Read-only?	Yes

Cluster Configuration

You can configure the following cluster components through metadata.

- ReplicationSyncserver
- MessagingSyncserver

- AdministrationListener
- SecureAdministrationListener
- HTTPListener
- SecureHTTPListener
- SSLSecurityProfile
- KeyStore
- TrustStore
- OCSP
- SolutionManager
- DCN Performance
- WebContainer
- ConfigurationCache

ReplicationSyncServer

The `ReplicationSyncServer` component contains the following configurable properties:

Table 246. ml.cachesize

Datatype	String
Default	50M
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 247. ml.threadcount

Datatype	int
Default	5
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 248. sup.sync.certificate

Datatype	String
Required?	Yes
Requires server restart?	Yes

Read-only?	No

Table 249. sup.sync.certificate_password

Datatype	String
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 250. sup.sync.httpsport

Datatype	int
Default	2481
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 251. sup.sync.port

Datatype	int
Default	2480
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 252. sup.sync.protocol

Datatype	String
Allowable values	• http • https
Default	http
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 253. sup.user.options

Datatype	String
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 254. sup.sync.e2ee_type

Datatype	String
Allowable values	• rsa • <empty>
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 255. sup.sync.e2ee_private_key

Datatype	String
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 256. sup.sync.e2ee_private_key_password

Datatype	String
Required?	No
Requires server restart?	Yes
Read-only?	No

MessagingSyncServer

The `MessagingSyncServer` component contains the following configurable properties.

Table 257. msg.admin.webservices.port

Datatype	int
Default	5100

Required?	No
Requires server restart?	Yes
Read-only?	No

Table 258. msg.http.server.ports

Datatype	String
Default	5001,80
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 259. sup.msg.inbound_count

Datatype	int
Default	50
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 260. sup.msg.inbound_queue_prefix

Datatype	String
Default	sup.mbs.
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 261. sup.msg.outbound_count

Datatype	int
Default	5
Required?	No
Requires server restart?	Yes

Read-only?	No

Table 262. sup.msg.outbound_queue_prefix

Datatype	String
Default	sup.mbs.moca.
Required?	No
Requires server restart?	Yes
Read-only?	No

AdministrationListener

The `AdministrationListener` component contains the following configurable properties:

Table 263. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 264. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 265. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes

Read-only?	No

Table 266. 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 267. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 268. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 269. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes

Read-only?	No

Table 270. sup.socket.listener.security.profile

Datatype	String
Default	default
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 271. 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 272. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 273. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes

Read-only?	No

Table 274. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 275. 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 276. sup.socket.listener.enabled

Datatype	boolean
Default	TRUE
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 277. sup.socket.listener.port

Datatype	int
Default	null
Required?	Yes
Requires server restart?	Yes

Read-only?	No

Table 278. sup.socket.listener.protocol

Datatype	String
Default	iiop
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 279. sup.socket.listener.security.profile

Datatype	String
Default	default
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 280. 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 281. 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 282. sup.security.profile.key.alias

Datatype	String
Default	null
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 283. 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 284. sup.sync.sslkeystore

Datatype	String
Default	Repository/Security/keystore.jks
Required?	Yes

Requires server restart?	Yes
Read-only?	Yes

Table 285. 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 286. sup.sync.ssltruststore

Datatype	String
Default	Repository/Security/truststore.jks
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 287. sup.sync.ssltruststore_password

Datatype	String
Default	changeit
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

OCSP

The `OCSP (Online Certificate Status Protocol)` component contains the following configurable properties:

Table 288. ocsp.enable

Datatype	Boolean

Default	False
Required?	No
Requires server restart?	No
Read-only?	No

Table 289. ocsp.responderURL

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Table 290. ocsp.responderCertIssuerName

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Table 291. ocsp.responderCertSerialNumber

Datatype	String
Default	None
Required?	No
Requires server restart?	No
Read-only?	No

Table 292. ocsp.responderCertSubjectName

Datatype	String
Default	None
Required?	No

Requires server restart?	No
Read-only?	No

SolutionManager

The `SolutionManager` component contains the following configurable properties.

Table 293. com.sap.solutionmanager.url

Datatype	string
Default	
Required?	Yes
Requires Server Restart?	Yes
Read-only?	No

DCN

The `DCN` component contains the following configurable properties.

Table 294. sup.dcn.http.get.enabled

Datatype	boolean
Default	False
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Performance

The `Performance` component contains the following configurable properties.

Table 295. sup.msg.inbound_count

Datatype	int
Default	25
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 296. sup.msg.outbound_count

Datatype	int
Default	5
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 297. sup.msg.subscribe.receiver.count

Datatype	int
Default	5
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 298. ml.threadcount

Datatype	int
Default	10
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 299. ml.cachesize

Datatype	string
Default	70p
Required?	Yes
Requires server restart?	Yes
Read-only	No

Table 300. webservices.connections.max.total

Datatype	int
Default	250

Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 301. webservices.connections.max.per.host

Datatype	int
Default	50
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 302. webservices.connection.manager.timeout

Datatype	long
Default	180000
Required?	Yes
Requires server restart?	Yes
Read-only?	No

WebContainer

The `WebContainer` component contains the following configurable properties.

Table 303. gzipFilter

Datatype	boolean
Default	False
Required?	No
Requires server restart?	Yes
Read-only?	No

Table 304. maxFormContentSize

Datatype	string
Default	200000

Required?	No
Requires server restart?	Yes
Read-only?	No

ConfigurationCache

The `ConfigurationCache` component contains the following configurable properties.

Table 305. cache.core.pool.size

Datatype	int
Default	40
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 306. cache.max.pool.size

Datatype	int
Default	40
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 307. cache.keep.alive.time

Datatype	int
Default	300
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 308. cache.network.encrypt

Datatype	boolean
Default	False

Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 309. cache.port.autoinc

Datatypes	boolean
Default	False
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 310. cache.port

Datatype	int
Default	5701
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Server Configuration

You can configure the following components through metadata:

- ConsolidatedDB
- JVM

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.

ConsolidatedDB

The `ConsolidatedDB` component contains the following configurable properties:

Table 311. cdb.asa.mode

Datatype	String
Default	primary
Required?	No

Requires server restart?	Yes
Read-only?	Yes

Table 312. cdb.databasename

Datatype	String
Default	default
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 313. cdb.dnsname

Datatype	String
Default	default-cdb
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 314. cdb.install_type

Datatype	String
Default	default
Required?	No
Requires server restart?	Yes
Read-only?	Yes

Table 315. cdb.password

Datatype	String
Default	sql
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 316. cdb.serverhost

Datatype	String
Default	gma
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 317. cdb.servername

Datatype	String
Default	gma_primary
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 318. cdb.serverport

Datatype	int
Default	5200
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 319. cdb.threadcount

Datatype	int
Default	20
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 320. cdb.type

Datatype	String

Allowable values	• Sybase_ASA • Sybase_ASE
Default	Sybase_ASA
Required?	Yes
Requires server restart?	Yes
Read-only?	Yes

Table 321. cdb.user.options

Datatype	String
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 322. cdb.username

Datatype	String
Default	dba
Required?	Yes
Requires server restart?	Yes
Read-only?	No

JVM

The `JVM` component contains the following configurable properties:

Table 323. DJC_JVM_MINHEAP

Datatype	String
Default	64M
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 324. DJC_JVM_MAXHEAP

Datatype	String
Default	256M
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 325. DJC_JVM_STACKSIZE

Datatype	String
Default	400K
Required?	Yes
Requires server restart?	Yes
Read-only?	No

Table 326. DJC_JVM_USEROPTIONS

Datatype	String
Default	""
Required?	No
Requires server restart?	Yes
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 327. LogLevel

Datatype	String
Allowable values	• TRACE • DEBUG • INFO • WARN • ERROR • OFF
Default	WARN
Required?	No
Requires server restart?	No
Read-only?	No

Table 328. async

Datatype	boolean
Default	FALSE
Required?	No
Requires Server Re-start?	No
Read Only?	No

Table 329. dateRollover

Datatype	String
Allowable Values	• NONE • HOURLY • DAILY • WEEKLY • MONTHLY • YEARLY

Default	NONE
Required?	No
Requires Server Restart?	No
Read Only?	No

Table 330. filename

Datatype	String
Default	null
Required?	Yes
Requires Server Restart?	No
Read Only?	No

Table 331. maximumRolloverFiles

Datatype	int
Default	1
Required?	No
Requires Server Restart?	No
Read Only?	No

Table 332. sizeRollover

Datatype	String
Default	10mb
Required?	No
Requires Server Restart?	No
Read Only?	No

Property Reference

Review properties of the Management 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.	
Customization Resource	The application configuration (customization resource bundles) associated with the application.	

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
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
BES Push Listener Port	The listener port for BES notifications. The port is discovered and set by the client, and is read-only on the server.
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
BES Notification 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 for a client application so it can locate the appropriate Unwired Server synchronization service.

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

ID: property name (type)	Description	Default
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.	5011
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
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 Hybrid App-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
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: Basic information, including errors • 2: Some additional details beyond basic • 3: Medium amount of information logged • 4: Maximum tracing of debugging and timing information • 5: Maximum tracing of debugging and timing information (currently same as level 4)	1

ID: property name (type)	Description	Default
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. For IIS, this path is relative. Acceptable values include: • For IIS – use `/ias_relay_server/client/rs_client.dll`. • For Apache – use `/cli/iasrelayserver`. **Note:** The value used in the client application connection for the URL suffix must match what the administrator configures in the URL suffix. Otherwise, the connection fails. Use the Diagnostic Tool command line utility to test these values. See *Diagnostic Tool Command Line Utility (diagtool.exe) Reference* in *System Administration*.	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_host-name>:8000/GWC/SUP-Notification

Security Settings Properties

Security settings display the device security configuration.

ID: property name (type)	Description	Default
E2E Encryption Enabled	Allows you to indicate whether end-to-end encryption is enabled or not: true indicates encryption is enabled; false indicates encryption is disabled.	
E2E Encryption Type	Allows you to use RSA as the asymmetric cipher used for key exchange for end-to-end encryption.	
TLS Type	Allows you to use RSA as the TLS type for device to Unwired Server communication.	

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
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:

```
jdbc:<NativeConnPropName>=<Value>
```

Note: If Unwired Server is connecting to a database with a JDBC driver, ensure you have copied required JAR files to correct locations. See *Installation Guide for Runtime*.

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.
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 aliasFor 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: `SUP_HOME\Servers\SQLAny-where11\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: `SUP_HOME\Servers\SQLAnywhere11\BIN32\dbsrvXX.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 SQL Anywhere® database, where the user name and password are the defaults (dba and sql), enter:
		
		`SUP_HOME\Servers\SQLAny-where11\BIN32\dbsrvXX.exe`

databaseType	Specifies the database type.	`<database type>`

Name	Description	Supported values
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 *`SUP_HOME`*`\lib\jdbc`. Set isDownloadZipped to true to save the file to *`SUP_HOME`*`\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. The default is 10.
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: `${selectList}${into-Clause}${fromClause} holdlock${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)*.

This list of properties can be used by all datasource types. Sybase does not document all native endpoint properties. However, you can add native endpoint properties, naming them using this syntax:

```
<NativeConnPropName>=<SupportedValue>
```

Table 333. General connection parameters

Name	Description	Supported values
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.	Not supported. Do not set this parameter or leave it set to 0.
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.	Not supported. Do not set this parameter or leave it set to 0.
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.	Not accessible by the customer. Do not set this parameter or leave it set to 0.

Name	Description	Supported values
jco.client.x509cert	Unwired Platform sets this property when a client uses an X509 certificate as the login credential.	If an EIS RFC operation is flagged for SSO (user name and password personalization keys selected in the authentication parameters) then Sybase Unwired Platform automatically sets the appropriate properties to use X.509, SSO2, or user name and password SSO credentials. The corresponding properties should not be set by the administrator on the SAP endpoint.
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.	Not supported.
jco.client.use_guihost	Identifies which host to redirect the remote graphical user interface to.	Not supported.
jco.client.use_guiserv	Identifies which service to redirect the remote graphical user interface to.	Not supported.
jco.client.use_guiprogid	Indicates the program ID of the server that starts the remote graphical user interface.	Not supported.

SAP DOE-C Properties

Configure Sybase 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 Sybase SAP® Data Orchestration Engine Connector 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.
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).

Name	Description	Supported values
doe-extract-window	Specifies the number of messages allowed in the DOE extract window.	Positive value (in messages). The minimum value is 10. The maximum value is 2000. 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.
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.	Positive value (in bytes). The default is 1048576 bytes (1MB). Do not set lower than 4096 bytes; there is no maximum limitation.
service-address	Specifies the DOE URL.	Valid DOE URL. If you are using DOE-C with SSL: • 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 endpoint 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://<sup_hostname>:8000/doe/publish`.

Name	Description	Supported values
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.
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

Name	Description	Supported Values
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.
authentication-Preemptive	When credentials are available and this property is set to the default of false, this property allows Unwired Server to send the authentication credentials only in response to the receipt of a server message in which the HTTP status is 401 (UNAUTHORIZED) and the WWW-Authenticate header is set. In this case, the message exchange pattern is: request, UNAUTHORIZED response, request with credentials, service response. When set to true and basic credentials are available, this property allows Unwired Server to send the authentication credentials in the original SOAP or REST HTTP request message. The message exchange pattern is: request with credentials, a service response.	False (default) True
Socket Timeout	The socket timeout value controls the maximum time in milliseconds after a web service operation (REST or SOAP) is allowed to wait for a response from the remote system; if the EIS system doesn't respond in that time, the operation fails and the SUP thread is unblocked.	Time in milliseconds (default: 6000). Range of [0 – 2147483647], where 0 is interpreted as infinity.

Proxy Endpoint Properties

Configure connection properties for the SAP Gateway proxy connection.

Name	Description	Supported values
password	Specifies the password for authentication.	Password
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}).

Numeric Error Code	Message
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.
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.

Numeric Error Code	Message
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}).
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}).

Numeric Error Code	Message
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 (${reason_sub_2}) is not supported.
00052	Failed to export package (${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}).

Numeric Error Code	Message
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 (${reason_sub_2}) is not supported.
00072	Failed to retrieve the synchronization group (${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.

Numeric Error Code	Message
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}).
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}).

Numeric Error Code	Message
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 (${reason_sub_2}) is not supported.
00116	Failed to connect to the configured server connection (${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.

Numeric Error Code	Message
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 (${reason_sub}) is missing.
00138	Failed to retrieve monitoring data (${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}).

Numeric Error Code	Message
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 (${reason_sub}) to the server does not exist.
00160	Failed to stop Unwired Server (${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.

Numeric Error Code	Message
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.
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.

Numeric Error Code	Message
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.
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}).

Numeric Error Code	Message
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 (${error_sub}). A monitored target is required.
00212	The value entered exceeds the maximum value allowed (${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.

Numeric Error Code	Message
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 (${error_sub}) retrieve failed.
00232	User (${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.

Numeric Error Code	Message
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 (${error_sub}) update failed.
00251	Messaging-based synchronization server apple push notification certificate (${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.

Numeric Error Code	Message
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 (${reason_sub}) is in progress.
00269	Cannot deploy the package as (${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.

Numeric Error Code	Message
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.
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.
00297	Failed to do the operation, because the login user doesn’t have authorization to do the operation.

Backward Compatibility

When upgrading from a previous version of Sybase Unwired Platform, certain APIs are no longer supported, or are supported with limitations.

Note: After upgrading the Unwired Server to release 2.2, replace `sup-mms-admin-api-lite.jar` with the new version of that file because the underlying implementation has changed.

Note: `SUPServerLog` settings are now configured at the cluster level but are still compatible with previous versions.

These APIs are no longer supported:

- `SUPMobileWorkflow`: all methods of this class throw an UnsupportedOperationException when called. Use the new`SUPMobileHybridApp` API instead.
- `SUPDeviceUser`: all methods of this class throw an UnsupportedOperationException when called.
- `SUPApplication`: the following methods of this class throw an exception when called.
  - getApplicationConnectionTemplateSettings
  - getApplicationConnectionSettings
  - createApplicationConnectionTemplate
- `SUPServerConfiguration`: the following methods of this class throw an UnsupportedOperationException when called.
  - getReplicationSyncServerConfiguration
  - updateReplicationSyncServerConfiguration
  - getMessagingSyncServerConfiguration
  - updateMessagingSyncServerConfiguration
  - getConsolidatedDatabaseConfiguration
  - getAdministrationListenerConfiguration
  - updateAdministrationListenerConfiguration
  - getHTTPListenerConfigurations
  - addHTTPListenerConfiguration
  - deleteHTTPListenerConfiguration
  - updateHTTPListenerConfiguration
  - getSecureHTTPListenerConfigurations
  - addSecureHTTPListenerConfiguration
  - deleteSecureHTTPListenerConfiguration
  - updateSecureHTTPListenerConfiguration

- getSSLSecurityProfileConfigurations
- addSSLSecurityProfileConfiguration
- deleteSSLSecurityProfileConfiguration
- updateSSLSecurityProfileConfiguration
- getKeyStoreConfiguration
- updateKeyStoreConfiguration
- getTrustStoreConfiguration
- updateTrustStoreConfiguration
- getApplePushNotificationConfigurations
- addApplePushNotificationConfiguration
- deleteApplePushNotificationConfiguration
- updateApplePushNotificationConfiguration

These APIs are supported with limitations:

Table 334. 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 335. 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.

Notification API

You can use the Notification API to make requests from the EIS for Sybase Unwired Platform to send push notifications to devices. The Notification API is an HTTP API that consists of a URL and, depending on notification mode may contain HTTP headers and a body. The header fields send notification data from the backend as a generic HTTP header for any device, or as device-specific HTTP headers.

Notification Mode

The notification mode specifies how native notifications or payload push notifications to registered devices are delivered. You select the notification mode when you create an application connection.

- Only native notifications – allows third party applications or EIS to deliver native notifications directly through the HTTP notification channel: BlackBerry (BES), Apple (APNS), or Android (GCM) to the device.

  In this notification mode, you send notification data through the request headers. There is no body in this notification mode.
- Only online/payload push – allows third party applications or EIS to deliver data payload push notifications to an online device.

  In this notification mode, no headers are required, and you send the business payload through the HTTP body.
- Online/payload push with native notifications – allows both payload and native notifications to be delivered to the device.

  In this notification mode, no headers are required, and you send the business payload through the HTTP body.

Note: Apple and Google do not recommend payload delivery over their systems:

- Data may not be delivered.
- Data is delivered out of sequence.
- If enabled, the actual payloads must be small.

  For example, GCM makes no guarantees about delivery or order of messages. While you might use this feature to inform an instant messaging application that the user has new messages, you probably would not use it to pass actual messages.

RIM supports guaranteed delivery, including callbacks to notify Unwired Server that the message was delivered or when it failed. However, this is a different message format. RIM messaging has many limitations including packet size, number of packets for a single user that BES keeps, number of packets BES keeps for all users, and so on. Therefore, BES should also only be used to send the notification, but not to send payloads.

Refer to the appropriate platform documentation (APNS, BES, or GCM) for additional information.

Notification URL

Send a push notification request from the EIS using an HTTP POST to the push notification URL.

```
 http://{unwired_server_name}:{unwired_server_port}/notifications/
v1/{appcid}
```

Parameters:

- **appcid –** The application connection ID of the application instance interacting with the service.

Note: You can also send the Application Connection ID as the "x-sap-push-deviceID" header.

Notification Headers

You can send notification data from the backend as a generic HTTP header for any device, or as device-specific HTTP headers.

Apple Header Fields

The structure of HTTP headers for sending Apple Push Notification Service (APNS) notifications through Sybase Unwired Platform.

- **< x-sup-apns-alert> –** The alert can consist of either a JSON-formatted request, or as these separate headers:
  - **<x-sup-apns-alert-body> –** The text of the alert message.
  - **<x-sup-apns-alert-action-loc-key> –** If a string is specified, displays an alert with two buttons, Close and View. iOS uses the string as a key to get a localized string in the current localization to use for the right button's title instead of "View." If the value is null, the system displays an alert with a single OK button that dismisses the alert when tapped.
  - **<x-sup-apns-alert-loc-key> –** A key to an alert-message string in a `Localizable.strings` file for the current localization.
  - **<x-sup-apns-alert-loc-args> –** Variable string values to appear in place of format specifiers in `<x-sup-apns-alert-action-loc-key>`.
  - **<x-sup-apns-alert-launch-image> –** The filename of an image file in the application bundle; it may include the extension or omit it. The image is used as the launch image when users tap the action button or move the action slider. If this property is not specified, the system either uses the previous snapshot, uses the image identified by the

`UILaunchImageFile` key in the application's `Info.plist` file, or falls back to `Default.png`. This property was added in iOS 4.0.

- **<x-sup-apns-badge> –** Number to be displayed as the badge on the application icon.
- **<x-sup-apns-sound> –** Name of the sound file in the application bundle.
- **<x-sup-apns-data> –** Custom payload data values that must use the JSON structured and primitive types: dictionary (object), array, string, number, and Boolean.

For more information on APNS headers, see *http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html*

Google Header Fields

The list of HTTP headers to be used in the HTTP request for sending Google Cloud Messaging (GCM) notifications.

- **<x-sup-gcm-collapsekey> –** An arbitrary string (such as "Updates Available") that is used to collapse a group of like messages when the device is offline, so that only the last message gets sent to the client.
- **<x-sup-gcm-data> –** Payload data, expressed as parameters prefixed with data and suffixed as the key.
- **<x-sup-gcm-delaywhileidle> –** Optional. Requests that messages not be delivered until the device becomes active Represented as `1` for true, or any other value for false. The default value is false.
- **<x-sup-gcm-timetolive> –** Specifies how long (in seconds) the message should be kept on GCM storage if the device is offline.
- When you do not send the the collapse key, the default 'Collapse_Key' value is used, "Updates Available."

For more information on GCM headers, see *http://developer.android.com/guide/google/gcm/gcm.html#send-msg*

BlackBerry Header Fields

The list of HTTP headers to be used in the HTTP request for sending BlackBerry notifications.

- **<x-sup-rim-data> –** A binary value (base64 encoded).

Generic Header Fields

You can use the generic header in the HTTP request to send any type of notification. Unless you require platform-specific features in your notification (for example, the display of a button with specific text), you can more easily use the generic header.

- **<x-sup-data> –** The notification data.

Sybase Unwired Platform interprets the header for each device type as follows:

- APNS: custom payload data values that use the JSON structured and primitive types: dictionary (object), array, string, number, and Boolean.
- GCM: payload data, expressed as parameters prefixed with data and suffixed as the key.
- Blackberry: a binary value (base64 encoded).

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

R

S

T

U

V