Developer Guide: OData SDK

SAP Mobile Platform 2.3 SP03

Contents

CHAPTER 1 Overview

The SAP® Mobile Platform provides two sets of libraries to build online applications: OData SDK and REST SDK. It is recommended to use the REST SDK for building online applications. You also have an option to enable mixed connectivity using OData SDK and REST SDK.

OData for SAP Products

OData stands for "Open Data Protocol" and is a resource-based web protocol for querying and updating data. It is released by Microsoft under the Open Specification Promise to allow anyone to freely interoperate with OData implementations. OData defines operations on resources using HTTP verbs (GET, PUT, POST, and DELETE), and it identifies those resources using a standard URI syntax. Data is transferred over HTTP using the Atom or JSON format.

OData for SAP® Products provide SAP Extensions to the OData protocol that enable users to build user interfaces for accessing the data published via OData. The interfaces require human-readable, language-dependent labels for all properties and free-text search within collections of similar entities and across (OpenSearch).

Applications running on mobile devices also require semantic annotations to tell the client which of the OData properties contain a phone number, a part of a name or address, or something related to a calendar event, thus seamlessly integrating with the contacts, calendar, and telephony of the mobile device. The OData standard's metadata document contains information about the model. It will define what information is searchable, which properties may be used in filter expressions, and which properties of an entity will always be managed by the server.

For the sake of simplicity, "OData for SAP" is abbreviated to "OData" throughout this Guide.

Related Information

For supporting information, see:

- *Developer Guide: Migrating to SAP Mobile SDK.*
- *Supported Hardware and Software*
- *New Features.*

OData SDK Components — General Description

The different components of the OData SDK are implemented as static runtime libraries and each component can be used independently.

The OData SDK is used for building native mobile applications. It consists of a collection of runtime libraries and classes. The OData SDK supports Android, BlackBerry and iOS platforms and it is based on the native device SDKs of the platforms. There is an implementation for each platform. Native applications installed on the devices allow the client application to leverage the support provided by the given platform, for example:

- Adapt to each device's form factor (for example, automatic layout)
- Exploit different input methods (for example, touch screen, keyboard or trackball)
- Cache data in native device data stores for better performance
- Tightly integrate with the features of the device

The general description of the SDK components follows. For detailed platform specific descriptions, see the respective chapters on Android, BlackBerry and iOS.

Note: Before using SAP® Mobile Platform 2.2 SDK, the OData applications built on 2.1 ESD # 3 or prior releases should to be explicitly migrated using the migration procedures in the *Developer Guide: Migrating to SAP Mobile SDK* guide for 2.2. When you add the SAP Mobile Platform 2.2 OData SDK into your existing application, the application will encounter compilation errors, that should be resolved. This migration is not required for releases 2.2 and above.

The following components are included in the OData SDK. See the detailed platform specific descriptions in the respective sections.

OData Parser

Parses and generates valid OData Protocol messages to/from native objects. It eliminates the need for mobile developers to work with the low-level details of the OData protocol directly. Functionalities supported by this component include:

- Parsing OData XML structures to native OData objects
- Validating OData XML during parsing by checking the existence of mandatory fields and structures
- Providing easy access to all OData fields and structures via the objects resulting from the parsing
- Building OData XML structures from native OData objects

Cache Management

The runtime cache is responsible for storing and accessing OData related objects in the memory of the device for quick and easy access. Functionalities supported by this component include:

- Storing/accessing OData objects in the memory (both metadata and application data)
- Searching for OData entries in memory using their searchable fields
- Managing the size of the cache

Persistence
Implements a convenient and secure storage of data on the device. Mobile applications can access the locally stored data even when network connection is unavailable. Functionalities supported by this component include:

- Storing objects and raw data on the physical storage of the device
- Easy and quick access of the stored objects and raw data
- Data encryption for sensitive data

Supportability
Implements standard SAP logging, tracing and error handling to enable end-to-end supportability from client to back-end. Functionalities supported by this component include:

- Common exception and error handling
- Event logging
- Tracing (SAP Passport)

Connectivity
This connectivity layer handles all connectivity related tasks, hides the complexity of the communication at transport level, and provides an easy to use API to the applications. Productive enterprise applications must use SAP Mobile Platform for connectivity for all enterprise use cases. For development and demo purposes, the SDK also provides a possibility to use HTTP or HTTPS. Functionalities supported by this component include:

- Synchronous and asynchronous HTTP request handling
- Supported authentication are:
  - X509 certification
  - SSO token/cookie
  - Basic authentication (user/password)
- Timeout handling
- Compressed payload handling
- Request types as supported by OData Protocol
- Connection pools for optimal performance

REST SDK Components — General Description

(Applicable to iOS and Android platforms) The REST SDK libraries enable consumption of SAP Mobile Platform REST services with pure HTTP (by default in on-premise) connectivity. The REST SDK provides simplified APIs for registration, exchange settings between client and server, and end-to-end tracing. The SDK also supports native push notifications.

The different components of the REST SDK are implemented as static runtime libraries and each component can be used independently.

The REST SDK supports the following functionalities:

- Initialize client connection
- User on-boarding or registration
- Application connection setting exchange
- Customization resource bundle
- Security configurations - Basic authentication, SSO certificate authentication, anonymous access, network edge (SiteMinder authentication - always asynchronous), non-network edge, single SSL and mutual SSL authentication
- End-to-end tracing
- Native notification support for iOS and Android platforms
- CAPTCHA support in cloud version. Both synchronous and asynchronous requests are supported. But asynchronous is recommended during on-boarding using CAPTCHA in productive scenario.
- Non network edge authentication support for iOS and Android platforms. In Network edge (siteminder) scenario, relay server is configured. User needs to get authenticated and moves for authentication to server and then to backend.

Feature differences between On-Premise and Cloud versions

Differences between On-Premise and Cloud functionality.
The table provides the list of features supported in On-Premise and/or Cloud versions.

Feature Support	On-Premise	Cloud
Notification of Customization resource bundle	No	Yes
Application connection setting exchange. **Note:** Application connection setting exchange means whether the server supports accessing services without registration.	No	Yes
CAPTCHA support (asynchronous)	No	Yes
BTX upload	Yes	No
Resource bundles configuration at application connection level	Yes (only at application connection level)	Yes

Feature Support	On-Premise	Cloud
Mutual SSL authentication (between client and SAP Mobile Platform server)	Yes	No
Network edge authentication	Yes	No
HTTP **Note:** The default connectivity is HTTPS.	Yes	No
End-to-end tracing	Yes	No

Mixed Connectivity Using OData SDK and REST SDK

The existing applications from release 2.2 SP02 and earlier versions can make use of both OData SDK and REST SDK.

For more information about how to enable mixed connectivity for iOS, see *Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)* on page 77.

For more information about how to enable mixed connectivity for Android, see *Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)* on page 227.

Documentation Roadmap for SAP Mobile Platform

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

CHAPTER 2 Developing iOS Applications

Provides information about using advanced SAP® Mobile Platform features to create applications for Apple iOS devices. The target audience is advanced developers who are familiar working with APIs, but who may be new to SAP Mobile Platform.

Describes requirements for developing a device application for the platform. Also included are task flows for the development options, procedures for setting up the development environment and API references.

1. *Getting Started Task Flow for iOS Applications*

   This task flow describes requirements for developing a device application for the platform. It includes task flows for the development options, procedures for setting up the development environment, and API documentation.
2. *Development Task Flow for iOS Applications Using OData SDK (Messaging Channel)*

   Describes the overall development task flow for ODP applications using messaging channel. An application consists of building blocks which the developer uses to start the application and perform functions needed for the application.
3. *Development Task Flow Using REST SDK (HTTP Channel)*

   Describes the overall development task flow for ODP applications using REST SDK. An application consists of building blocks which the developer uses to start the application and perform functions needed for the application.
4. *Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)*

   Use setRequestType method in SDMRequestBuilder class to set the request-response using Messaging Channel or REST SDK.
5. *Deploying Applications to Devices*

   Complete steps required to deploy mobile applications to devices.
6. *Testing Applications*

   Test native applications on a device or simulator.
7. *OData SDK Components and APIs*

   The iOS OData SDK provides the means to easily build an app which relies on the OData protocol and the additions made by SAP.

Getting Started Task Flow for iOS Applications

This task flow describes requirements for developing a device application for the platform. It includes task flows for the development options, procedures for setting up the development environment, and API documentation.

See also

- *Development Task Flow for iOS Applications Using OData SDK (Messaging Channel)* on page 12

Installing the iOS Development Environment

Install the iOS development environment, and prepare iOS devices for authentication.

Downloading the Xcode IDE

Download and install Xcode.

1. Download Xcode from the Apple Web site: *http://developer.apple.com/xcode/*.
2. Complete the Xcode installation following the instructions in the installer.

Downloading Older Versions of the Xcode IDE

If you do not have the supported version of Xcode and the iOS SDK, you need to download it from the Downloads for Apple Developers Web site.

See *OData SDK Applications—Supported Versions* in *Supported Hardware and Software* for the most current version information for mobile device platforms and third-party development environments. If necessary, you can download older versions.

1. Go to *http://developer.apple.com/downloads/*.

   You must be a paying member of the iOS Developer Program. Free members do not have access to the supported version.
2. Log in using your Apple Developer credentials.
3. (Optional) Deselect all Categories except Developer Tools to narrow the search scope.
4. Download the supported Xcode and SDK combination.

Setting Up the Development Environment

Set up the iOS development environment by downloading the required plugins.

Importing Libraries to a Project

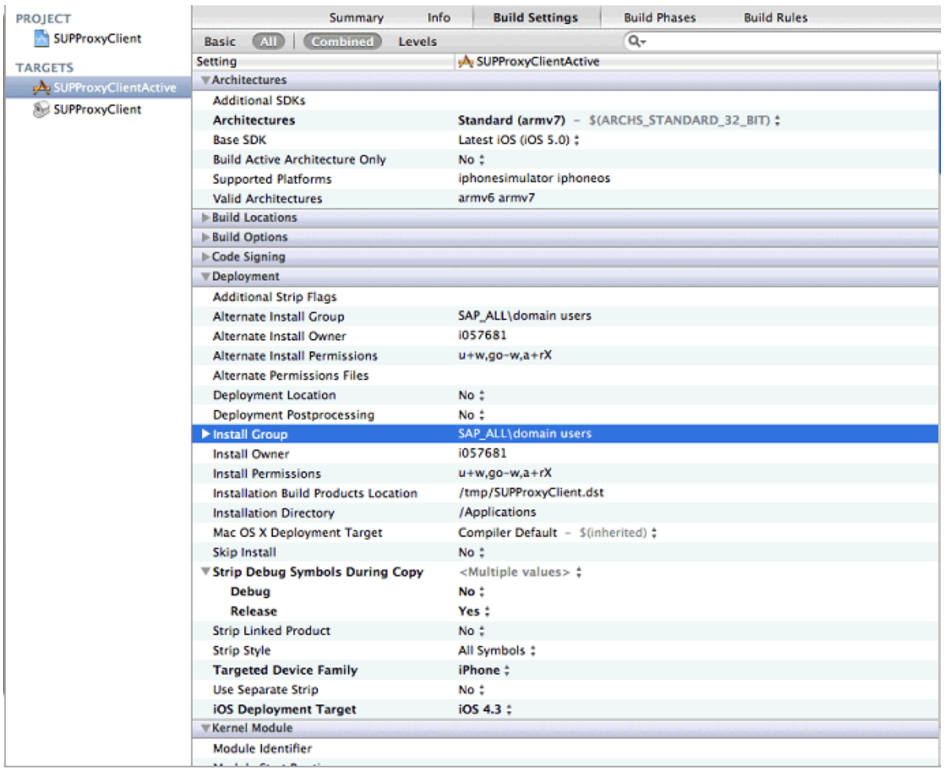
Import the associated iOS libraries into the iOS development environment.

Note: For more information on Xcode, refer to the Apple Developer Connection: *http://developer.apple.com/tools/Xcode/*.

1. Start Xcode 4.3 and select **Create a new Xcode project**.

   If you are using Xcode 4.5 or 4.6, the Xcode build settings might vary. For example: The architecture will have argv7 and argv7s by default. You might get build errors if argv7s is not removed.
2. Under **iOS**, select **Applications**.
3. In the right pane, select **Empty Application** as the project template and click **Next**.
4. Enter `<ProjectName>` as the **Product Name**, `<CorpID>` as the **Company Identifier**, select **Universal** as the **Device Family** product, then click **Next**.
5. Select a location to save the project and click **Create** to open it.
6. Select the **Build Settings** tab and scroll to the **Architectures** section.
7. Set **Base SDK** for **All** configurations to `iOS 5.1`.
8. Select the Valid architecture as `armv6 armv7` , Supported Platforms as `iphonesimulator iphoneos` , and the Targeted device family as `iPhone/iPad`. This ensures that the build of the application can run on either iPhone or iPad.

   Note: When you migrate an existing project from an older version of Xcode to Xcode 4.3, you may see a build error: `No architectures to compile for (ARCHS=i386, VALID_ARCHS=armv6,armv7)`. You can resolve this Xcode 4 issue by manually editing "Valid Architectures" under Targets, to add `i386`.
9. Select the **Deployment** tab and set the iOS Deployment Target as appropriate for the device version where you will deploy. The minimum version is `iOS 4.3`. Earlier SDKs and deployment targets are not supported.



Setting	SUPProxyClientActive
▼Architectures	
Additional SDKs	
Architectures	**Standard (armv7)** – $(ARCHS_STANDARD_32_BIT)
Base SDK	Latest iOS (iOS 5.0)
Build Active Architecture Only	No
Supported Platforms	iphonesimulator iphoneos
Valid Architectures	armv6 armv7
▶Build Locations	
▶Build Options	
▶Code Signing	
▼Deployment	
Additional Strip Flags	
Alternate Install Group	SAP_ALL\domain users
Alternate Install Owner	i057681
Alternate Install Permissions	u+w,go-w,a+rX
Alternate Permissions Files	
Deployment Location	No
Deployment Postprocessing	No
▶Install Group	SAP_ALL\domain users
Install Owner	i057681
Install Permissions	u+w,go-w,a+rX
Installation Build Products Location	/tmp/SUPProxyClient.dst
Installation Directory	/Applications
Mac OS X Deployment Target	Compiler Default – $(inherited)
Skip Install	No
▼**Strip Debug Symbols During Copy**	<Multiple values>
Debug	**No**
Release	**Yes**
Strip Linked Product	No
Strip Style	All Symbols
Targeted Device Family	**iPhone**
Use Separate Strip	No
iOS Deployment Target	**iOS 4.3**
▼Kernel Module	
Module Identifier	

10. Connect to the Microsoft Windows machine where Mobile SDK is installed:

a) From the Apple Finder menu, select **Go > Connect to Server**.

b) Enter the machine name where the SDK is installed or IP address of the server, for example, `smb://<machine DNS name>` or `smb://<IP Address>`.

You see the shared directory.

11. Navigate to the *SMP_HOME*`\MobileSDK<Version>\OData\iOS\` directory in the SAP Mobile Platform installation directory, and copy the `includes` and `libraries` folders to the `<ProjectName>/<ProjectName>` directory on your Mac.

12. Right-click the `<ProjectName>` folder under the project, select **Add Files to "<ProjectName>"**, navigate to the `<ProjectName/ProjectName>/libraries/Debug-universal` directory, select all the libraries, unselect **Copy items into destination group's folder (if needed)**, and click **Add**.

The libraries are added to the project in the Project Navigator.

13. Click the project root and then, in the middle pane, click the **<ProjectName>** project.

a) In the right pane click the **Build Settings** tab, then scroll down to the **Search Paths** section.

b) Enter the location of your `includes` folder (`"$SRCROOT/<ProjectName>/includes/public/**"`) in the **Header Search Paths** field.

`$SRCROOT` is a macro that expands to the directory where the Xcode project file resides.

14. Set the Other Linker Flags to `-ObjC -all_load` for both the release and for the debug configuration. It is important that the casing of -ObjC is correct (upper case 'O' and upper case 'C'). Objective-C only generates one symbol per class. You must force the linker to load the members of the class. You can do this with the help of the -ObjC flag. You must also force inclusion of all your objects from your static library by adding the linker flag `-all_load`.
15. Add the following frameworks from the SDK to your project by clicking on the active target, and selecting **Build Phase** > **Link Binary With Libraries**. Click on the **+** button and select the following binaries from the list:
    - CoreFoundation.framework
    - QuartzCore.framework
    - Security.framework
    - libicucore.A.dylib
    - libstdc++.dylib
    - libz.1.2.5.dylib
    - CFNetwork.framework
    - MobileCoreServices.framework
    - SystemConfiguration.framework
    - MessageUI.framework
    - CFNetwork.framework
16. Select **Product > Clean** and then **Product > Build** to test the initial set up of the project. If you have correctly followed this procedure, you should receive a **Build Succeeded** message.

    To build a project using Automatic Reference Counting, under the **Build Settings** tab, set Automatic Reference Counting (ARC) to `YES`, if ARC option was un-checked while creating project.

Downloading the Latest Afaria Libraries

Afaria® provides provisioning of configuration data and certificates for your SAP Mobile Platform client application. Afaria libraries are packaged with SAP Mobile Platform, but may not be the latest software available. To ensure you have the latest Afaria libraries and header files, download Afaria software.

1. Navigate to the Mobile Enterprise Technical Support website at *http://frontline.sybase.com/support/downloads.aspx*.
2. If not registered, register for an account.
3. Log into your account.
4. Select Software Updates and download the latest Static Link Libraries.
5. Extract the contents of the downloaded zip file.

6. Copy the Afaria library and the `SeedDataAPI.h` and `SeedingAPISynchronous.h` header files into the iOS development environment.
7. Include the Afaria library and header files into your project..

Development Task Flow for iOS Applications Using OData SDK (Messaging Channel)

Describes the overall development task flow for ODP applications using messaging channel. An application consists of building blocks which the developer uses to start the application and perform functions needed for the application.

1. *Initializing the Application*

   Initialize the application when it starts the first time.
2. *Setting Server Details*

   Set or update the connection properties of the server before or after user registration.
3. *Enabling Single and Mutual SSL Authentication*

   Secure sockets layer (SSL) is a protocol that governs certificate authentication. It exchanges the certificate information. SSL also encrypts all information that flows between a client and a server.
4. *Registering a User*

   Using a predefined authentication mechanism, register a user automatically. You can register the user either synchronously or asynchronously.
5. *Sending Data Request to the Back-end*

   You can forward data request messages to the back-end through the SAP Mobile Platform server synchronously or asynchronously.
6. *Retrieving the Response from the Back-end*

   Based on the data request sent to the back-end, you need to retrieve the data as a response to the device.
7. *Using HTTPS over the SAP Mobile PlatformMessaging Channel*

   (Optional) Clients can securely communicate with the backend via the server through the HTTPS transport protocol. By using SSL over the SAP Mobile Platform Messaging Channel, all messages are encrypted and authenticated to ensure secure communication.
8. *Debugging Runtime Error and Performance Analysis*

   To handle occurrences of exceptions and special conditions that change the normal flow of the program execution, error handling has to be done appropriately.

See also

- *Getting Started Task Flow for iOS Applications* on page 8
- *Development Task Flow Using REST SDK (HTTP Channel)* on page 41

Initializing the Application

Initialize the application when it starts the first time.

The following code illustrates how to initialize an application.

```
ODPUserManager* userManager = [ODPUserManager
getInstance:@"com.sap.NewFlight"];
```

See also

- *Setting Server Details* on page 14

Downloading Customization Resource Bundles

(Optional) The application can download the custom resources using this API. The customized resource bundles are assigned to application connections in the SAP Control Center.

A customization resource bundle is a JAR file that includes a manifest file of name and version properties. It enables you to associate deployed client applications with different versions of customization resources. See *Application Customization Resource Bundles* in *SAP Control Center for SAP Mobile Platform* for information about the implementation task flow of customizations resource bundles.

The following code illustrates how to download customization resource bundles.

```
NSData* customResourceBundleData = [ODPAppSettings
getCustomizationResourceBundleWithCustomizationResource:CUSTOMRESOU
RCEBUNDLENAME userName:USERNAME passWord:PASSWORD error:&error];
  if (!error)
  {
      // Further processing
  }
```

Enabling Apple Push Notifications

You use the Apple Push Notification service to propagate information from the backend to the device. Using the native iOS push services , you can enable APNS for iOS applications.

The following code illustrates how to enable APNS on a device.

```
//Enable APNS
To enable APNS, you need to implement the following in the
application delegate of the application that has to receive
notifications.
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [ODPClientConnection setupForPush:application];
}
* Callback by the system where the token is provided to the client
application so that this
 can be passed on to the provider. In this case, "deviceTokenForPush"
and "setupForPush"
are APIs provided by SAP Mobile Platform to enable APNS and pass the
```

```
token to SAP Mobile Server.

- (void)application:(UIApplication *)app
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken {
    [ODPClientConnection deviceTokenForPush:app
deviceToken:deviceToken];
}
* Callback by the system if registering for remote notification
failed.
- (void)application:(UIApplication *)app
didFailToRegisterForRemoteNotificationsWithError:(NSError *)err {
    [ODPClientConnection pushRegistrationFailed:app errorInfo:err];
       }
// You can alternately implement the pushRegistrationFailed API:
// +(void)pushRegistrationFailed:(UIApplication*)application
errorInfo: (NSError *)err
* Callback when notification is sent.
- (void)application:(UIApplication *)application
didReceiveRemoteNotification:(NSDictionary *)userInfo {
    [ODPClientConnection pushNotification:application
notifyData:userInfo];
}
```

Setting Server Details

Set or update the connection properties of the server before or after user registration.

The following code illustrates how to set the server details.

```
NSError* settingsError = nil;
[userManager setConnectionProfileWithHost:@"relayserver.sybase.com"
port:5001 farm:@"sampleApp.FarmMBS" error:&settingsError];
```

See also

- *Initializing the Application* on page 13

Enabling Single and Mutual SSL Authentication

Secure sockets layer (SSL) is a protocol that governs certificate authentication. It exchanges the certificate information. SSL also encrypts all information that flows between a client and a server.

In single SSL connection, the client needs to trust the server certificate. This can be done one of the three ways:

- The CA certificate can be installed on the device trust store which the application takes while connecting to the HTTPS URL.
- The CA certificate can be bundled in the application and a client certificate can be sent in the onClientCertificateChallenge callback. This is not a recommended way, but can used in case of backward compatibility.
- If the application does not provide the certificate, application can implement the onCertificateChallenge listener. Where the application receives the server certificate, it

can choose to trust the certificate or not. Based on that it returns the boolean value as true or false.

In mutual SSL connection, two parties authenticate each other through verifying the provided digital certificate (P12), so that both parties are assured of the others' identity.

For the mutual SSL to be established, the server sends the certificate that is accepted on the client side in the -(void) onCertificateChallenge:(NSString*)certInfo delegate. The client sends the user certificate that should be authenticated by the server using onClientCertificateChallenge callback.

Examples

- **Single SSL and Mutual SSL –** The following sample code shows how to implement single and mutual SSL.

```
ODPUserManager *manager = [ODPUserManager
getInstance:<applicationID>];
[ODPUserManager enableHTTPS:YES];
    NSError *error=nil;
    [manager setConnectionProfileWithHost:<relayserverhost>
port:<relayport>
                                     farm:<relayserverfarmID>
error:&error];


    [ODPClientListeners
setHTTPSClientCertificateChallengeListenerDelegate:self];
    [ODPClientListeners
setCertificateChallengeListenerDelegate:self];
    [ODPClientListeners
setHTTPAuthChallengeListenerDelegate:self];
    BOOL flag = [manager registerUser:<username>
                       securityConfig:<securityconfiguration>
password:<password> error:nil
                           isSyncFlag:YES];

//Call back for the certificate challenge sent from the server
-(void) onCertificateChallenge:(NSString*)certInfo
{

    @try
    {
        NSLog(@"Cert Info is: %@", certInfo);
        [ODPClientListeners certificateChallengeResult:YES]; //
Certificate is accepted
    }
    @catch (NSException *exc)
    {
        NSLog(@"exception is %@",[exc description]);
    }
}

- (void) onClientCertificateChallenge
```

```
{
    SecIdentityRef outIdentity;

    outIdentity =[MutualSSLOverIMO getClientCertificate];
    [ODPClientListeners
httpsClientCertificateChallengeResult:outIdentity];


}

-(void)onClientCertificateChallengeResult:(SecIdentityRef)
identity
{
    NSLog(@" came for on client certificate challanege result");
}

-(void) onHTTPError:(int)code errorMessage:(NSString*)message
httpHeaders:(NSDictionary*)headers

{
    NSError *error;

    //Handle error scenario

}
//Read the contents of the certificates
+(SecIdentityRef)getClientCertificate
{
    identityApp = nil;

    NSString *thePath = [[NSBundle mainBundle]

pathForResource:<clientcertificate>ofType:@"p12"];
    NSData *PKCS12Data = [[NSData alloc]
                          initWithContentsOfFile:thePath];
    CFDataRef inPKCS12Data = (__bridge CFDataRef)PKCS12Data;
    CFStringRef password = CFSTR(“<p12certificatepassword>”);
    const void *keys[] = { kSecImportExportPassphrase };
    const void *values[] = { password };
    CFDictionaryRef options = CFDictionaryCreate(NULL, keys,
values, 1,
                                                 NULL, NULL);
    CFArrayRef items = CFArrayCreate(NULL, 0, 0, NULL);
    OSStatus securityError = SecPKCS12Import(inPKCS12Data,
options,
                                             &items);
    CFRelease(options);
    CFRelease(password);
    if (securityError == errSecSuccess) {
        NSLog(@"Success opening p12 certificate. Items: %ld",
              CFArrayGetCount(items));
        CFDictionaryRef identityDict =
CFArrayGetValueAtIndex(items, 0);
        identityApp =
        (SecIdentityRef)CFDictionaryGetValue(identityDict,
                                             kSecImportItemIdentity);
    } else {
        NSLog(@"Error opening Certificate.");
    }
    return identityApp;
}
```

Registering a User

Using a predefined authentication mechanism, register a user automatically. You can register the user either synchronously or asynchronously.

The following code illustrates how to register a user automatically:

```
[userManager registerUser:@"supuser123" securityConfig:@"SSO2Cookie"
password:@"s3puser" error:nil isSyncFlag:YES];
```

See also

- *Sending Data Request to the Back-end* on page 17

Implementing Asynchronous Delegates

When you register a user asynchronously, you need to implement asynchronous delegates.

The following code illustrates how to manually register a user asynchronously.

```
//As a prerequisite, ensure that the class which implements the
asynchronous call inherits the ODPUserManagerDelegate in its
corresponding .h file
//Example: @interface SUPProxyClientViewController :
UIViewController<ODPUserManagerDelegate>
[userManager setDelegate:self];
[userManager registerUser:@"<username>" securityConfig:@"SSO"
password:@"<password>" error:nil isSyncFlag:YES];

//Asynchronous Error Delegate Handling
-(void)userRegistrationFailed:(NSError*)error
{
//Implement user logic here
}

//Asynchronous Success Delegate Handling
-(void)userRegistrationSuccessful
{
//Implement user logic here
}
```

Sending Data Request to the Back-end

You can forward data request messages to the back-end through the SAP Mobile Platform server synchronously or asynchronously.

See also

- *Registering a User* on page 17
- *Retrieving the Response from the Back-end* on page 21

Creating a URL Request

Create a URL request that enables the device to forward a data request to the corresponding back-end.

The following code illustrates how to create a URL request.

```
NSString* urlString = [ODPAppSettings
getApplicationEndpointWithError:nil];
ODPRequest* request = [ODPRequest requestWithURL:[NSURL
URLWithString:urlString]];
/While using the ODP interface, you need to set the pre-processor
directive USE_MOBILE_OBJECTS along with USE_ODP_INTERFACE in the
Build Settings.
```

Enabling XCSRF Token Protection

For all modifying operations such as POST, PUT and DELETE, the XCSRF token is used in the HTTP request-response header field.

Once you have enabled XCSRF token on the device, the token is extracted after the first GET request triggered by the application. This token is retained in the device memory, and is added to all subsequent modifying requests.

The following code illustrates how to enable XCSRF token using `ODPRequest` class.

```
[ODPRequest enableXCSRF:YES];
```

The following code illustrates how to enable XCSRF token using `SDMRequestBuilder` class.

```
[SDMRequestBuilder enableXCSRF:YES];
```

Assigning Credentials

Assign the user credentials.

The following code illustrates how to assign credentials.

```
ODPRequest* request = [ODPRequest requestWithURL:[NSURL
URLWithString:urlString]]; [request setUsername:@"supuser123"];
[request setPassword:@"password"];
```

Adding Custom Headers

Add custom headers to a request message. This is a name/value pair that defines the operating parameters of an HTTP transaction. Custom headers are optional while sending a data request to the back-end.

The following code illustrates how to add custom headers to the request message.

```
 [request addRequestHeader:@"SET-COOKIE"
value:@"KKNBJDNADU23123SHSFH"];
```

Setting the Required Timeout

Set the timeout value up to which the application waits until the request reaches the server.

The following code illustrates how to set the timeout value.

```
[request setTimeOutSeconds:30];
```

Submitting the Request

Send the synchronous or asynchronous request message to the back-end.

The following code illustrates how to forward an asynchronous message.

```
[request startAsynchronous];
```

Usage

For an asynchronous request, you need to set the following delegates:

```
//Setting the Delegates
[request setDelegate: self];
[request setDidFailSelector:@selector(requestFailed:)];
[request setDidFinishSelector:@selector(requestSuccess:)];
```

Enabling Data Streaming

Streaming supports handling large data sent from device-to-server and vice-versa.

Streaming of response data obtained from the server on a client is the ability to receive data of unlimited size via a stream. The stream is processed as the data arrives and is not needed to be stored in the device memory at any point of time. The streaming can be done in both synchronous and asynchronous invocations of the client. The streaming does not impact the server.

Server to Device Streaming: To enable server to device streaming, the application has to implement the APIs in `SDMHttpRequestDelegate` class. The size of the streamed data chunks can be set by using the set method on the ODPRequest object.

Note: The maximum chunk size that can be set is up to 200KB. If chunk size is not set, 100KB is considered as default value.

```
//Invoke the following method when response header is available to
the application
(void)headersArrived:(NSMutableDictionary*)headers;


//Invoke the following method when a chunk of data is available to
the application
 (void)request:(ODPRequest*)request didReceiveData:(NSData*)data;


//Invoke the following method after all chunks of a response are
received and when a request is completed
 (void)requestFinished:(ODPRequest*)request;
```

```
//Invoke the following method to set the  data chunks size
[l_request setChunkSize:50000];
```

Device to Server Streaming: Application enables uploading data stream functionality by setting `shouldStreamPostDataFromDisk` to **YES**.

```
//Invoke the following method to enable streaming from device to
server
upload_request.shouldStreamPostDataFromDisk=YES;


//Invoke the following method to provide the absolute file path on
the device. This file path contains the upload data.
[upload_request setPostBodyFilePath:filePath];
```

```
Example for uploading data from device to server

     ODPRequest* upload_request = [ODPRequest requestWithURL:[NSURL
URLWithString:@"http://ldcigkq.wdf.sap.corp:50080/sap/opu/odata/
iwfnd/RMTSAMPLEFLIGHT/TravelagencyCollection"]];
     NSString *documentsDirectory = [NSHomeDirectory()
                   stringByAppendingPathComponent:@"Documents"];
     NSString *filePath = [documentsDirectory
                  stringByAppendingPathComponent:@"uploadData.xml"];
     [upload_request addRequestHeader:@"Content-Type"
value:@"application/atom+xml"];
     [upload_request setRequestMethod:@"POST"];
     [upload_request setPostBodyFilePath:filePath];
     upload_request.shouldStreamPostDataFromDisk=YES;
     upload_request.allowCompressedResponse=NO;
     [upload_request setUsername:@"supuser2"];
     [upload_request setPassword:@"s3puser"];
     [upload_request startSynchronous];

Example for downloading data from server to device

ODPRequest * l_request=[ODPRequest requestWithURL:[NSURL
URLWithString:@"http://vmw3815.wdf.sap.corp:50009/sap/opu/sdata/
iwfnd/RMTSAMPLEFLIGHT/FlightCollection"]];
    [l_request setAllowCompressedResponse:NO];
    [l_request setUsername:@"supuser"];
    [l_request setPassword:@"s3puser"];
    [l_request setChunkSize:100000];
    [l_request setDelegate:self];
    [l_request
setDidReceiveDataSelector:@selector(request:didReceiveData:)];
    [l_request
setDidReceiveResponseHeadersSelector:@selector(headersArrived:)];
    [l_request setDidFinishSelector:@selector(requestFinished:)];
    [l_request startSynchronous];
```

Retrieving the Response from the Back-end

Based on the data request sent to the back-end, you need to retrieve the data as a response to the device.

The following code illustrates how to retrieve the response from the back-end.

```
-(void)requestSuccess:(ODPRequest*)request
{
NSString* responseString = [request responseString];
NSData* responseData = [request responseData];
}
```

See also

- *Sending Data Request to the Back-end* on page 17

Using HTTPS over the SAP Mobile PlatformMessaging Channel

(Optional) Clients can securely communicate with the backend via the server through the HTTPS transport protocol. By using SSL over the SAP Mobile Platform Messaging Channel, all messages are encrypted and authenticated to ensure secure communication.

See also

- *Debugging Runtime Error and Performance Analysis* on page 25

Enabling HTTPS as a Transport Protocol

(Optional) You can set HTTPS as the transport protocol that the an OData client should use to communicate with a network edge (Example: Relay server).

Syntax

```
+(void)enableHTTPS:(BOOL)flag;
```

Parameters

- **flag –** Set to "true", if the protocol to be used is HTTPS.

  Set to "false", if the protocol to be used is HTTP.

  Note: The default protocol is HTTP.

Examples

- **Setting the Protocol**

  ```
  [ODPUserManager enableHTTPS:YES];
  ```

Enabling a Listener for HTTPS Support with Server Certificate Validation

(Optional) For a HTTPS protocol, when a client attempts to setup a trusted connection with the host (like a relay server), the host responds with a certificate. To verify if the server certificate is trusted or not, the application registers a delegate with the OData SDK. If the server certificate is trusted, the connection is successful. If the server certificate is not trusted, the delegate determines if the certificate should be deemed trusted and proceed with the connection.If no delegate is set, all certificate chains that are not automatically trusted are rejected.

See *Configuring SAP Mobile Server to use Relay Server* in *SAP Control Center for SAP Mobile Platform* for information on the configuration of relay server properties.

Syntax

To register a listener for certificate verification, implement the protocol `ODPCertificateChallengeListenerDelegate`

```
@protocol ODPCertificateChallengeListenerDelegate <NSObject>
@required
-(void) onCertificateChallenge:(NSString*) certInfo;
@end
```

Examples

- **Register a delegate to verify server certificate**

```
-(void) onCertificateChallenge:(NSString*)certInfo
{
NSLog(@"%@", certInfo);
[ODPClientListeners certificateChallengeResult:YES];
}

-(IBAction)httpsCheck:(UIButton*)button
{
@try
{
ODPUserManager* manager = [ODPUserManager
getInstance:@"NewFlight"];
[ODPClientListeners
setCertificateChallengeListenerDelegate:self];
[ODPUserManager enableHTTPS:YES];

NSError *error=nil;
[userManager setConnectionProfileWithHost:@"hostname" port:5001
farm:@"farm" error:&error;
[userManger registerUser:@"username"
securityConfig:@"securityConfig" password:@"password"
error:&error isSyncFlag:YES];
}
@catch (NSException *exception)
{
NSLog(@"%@", [exception callStackSymbols]);
NSLog(@"%@", [[exception userInfo] objectForKey:@"ErrorCode"]);
}
}
```

Enabling a Listener for HTTPS Support with Basic Authentication Challenge

When a client attempts to connect to a host (like a relay server), this connection is authenticated with a basic authentication challenge. To setup a basic authentication, the application registers a listener with OData SDK. If the `ODPHTTPAuthChallengeListenerDelegate` is not registered, an HTTP response code 401 is returned when a challenge is received.

See *Configuring SAP Mobile Server to use Relay Server* in *SAP Control Center for SAP Mobile Platform* for information on the configuration of relay server properties.

Syntax

To register a listener for certificate verification, implement the protocol `ODPHTTPAuthChallengeListenerDelegate`

```
@protocol ODPHTTPAuthChallengeListenerDelegate <NSObject>
@required
-(void) onHTTPAuthChallenge:(NSString*)host forUser:
(NSString*)userName withRealm:(NSString*)realm;
@end
```

Examples

- **Register a delegate to verify basic auth challenge**

```
 [ODPClientListeners setHTTPAuthChallengeListenerDelegate:self];
 [ODPUserManager enableHTTPS:NO];

 [userManager registerUser:@"usermanager"
securityConfig:@"HttpAuth" password:@"password" error:&error
isSyncFlag:true];
 -(void) onHTTPAuthChallenge:(NSString*)host forUser:
(NSString*)userName withRealm:(NSString*)realm
 {

 [ODPClientListeners httpAuthChallengeResult:YES
forUser:@"Mobile" withPassword:@"Ntwatch@123"];
 NSLog(@"%@--%@--%@", host, userName, realm);
 @try
   {
    NSString *host = @"relaymp.sap-ag.de"; NSInteger port = 443;
NSString *companyid = @"SSLOverIMOHTTPS";  NSError *error;
    ODPUserManager* userManager = [ODPUserManager
getInstance:@"com.sap.NewFlight"];
    [userManager setConnectionProfileWithHost:host port:port
farm:companyid error:nil];
    [ODPClientListeners
setHTTPAuthChallengeListenerDelegate:self];
    [ODPUserManager enableHTTPS:YES];
    [userManager registerUser:@"smpuser2"
securityConfig:@"HttpAuth" password:@"s3puser" error:&error
    isSyncFlag:true];

    }

  }
```

Registering a Listener for HTTP Errors in iOS Applications

To ensure that OData iOS clients are notified of HTTP errors while establishing a connection with the network edge, implement a listener.

Syntax

To register a listener for HTTP error codes, implement the protocol `ODPHTTPErrorListenerDelegate`.

```
@protocol ODPHTTPErrorListenerDelegate <NSObject>
@required
- (void) onHTTPError: (int)code errorMessage:(NSString*)message
httpHeaders:(NSDictionary*)headers;
```

Examples

- **Register a delegate for HTTP authentication challenge**

```
-(void) onHTTPError:(int)code errorMessage:(NSSTring*)message
httpHeaders:(NSDictionary*)headers
{
  if (code==xxx){
   //Display Error Messages
  }
}

-(IBAction)registerUser:(id)sender
{
 ODPUserManager* userManager = nil;
 @try
 {
  userManager = [ODPUserManager getInstance:@"NewFlight"];
  [ODPClientListeners setHTTPErrorListenerDelegate:self];
  [userManager setConnectionProfile:@<host> withSupPort:<port>
withServerFarmID:@<farmID>];
  [userManager registerUser:@<supuser>
withSecurityConfig:@<securityconfig> withPassword:@<password>];
 }
 @catch (NSException *exception {
  NSLog(@"%@:%@", [[exception userInfo]
objectForKey:@"ErrorCode"],[[exception userInfo]
objectForKey:@"ErrorMessage"]);
 }
}
```

Debugging Runtime Error and Performance Analysis

To handle occurrences of exceptions and special conditions that change the normal flow of the program execution, error handling has to be done appropriately.

The following code illustrates how you handle errors for asynchronous requests.

```
-(void)requestFailed:(ODPRequest*)request
{NSError* error = [request error];
NSInteger errorCode = [error code];
NSString* errorDesc = [error description];
}
```

See also

- *Using HTTPS over the SAP Mobile PlatformMessaging Channel* on page 21

End to End Tracing

End to end tracing enables an application developer and end user to trace a request that is sent from the client to the back-end. This spans the entire landscape where you can derive a correlation of traces at the client, server and back-end.

These correlated traces help in performance analysis and are centrally monitored on SAP Solution Manager. These are displayed as reports where you can extract information on failure of delivering a request, time taken for a request to reach a component and so on.

On the client side, the client framework enables an application developer to switch on the trace for messages. The client traces the request at predefined points and all these transactions/ requests are recorded in a Business Transaction XML. Additionally, the client maintains a unique identifier in the HTTP header called the SAP Passport that is used to correlate traces across various components. This Business Transaction XML can later be uploaded to the SAP Solution Manager which is a central location to correlate all logging information.

Using Tracing APIs in SUPProxyClient and SDMConnectivity Libraries

One way of enabling end to end tracing is by using supportability-related APIs defined in the SUPProxyClient library.

These APIs are available in the `ODPRequest` class.

Syntax

- Use the `setTraceLevel` API to set the passport trace level. This API belongs to the `SDMRequestBuilder` class which is part of the `SDMConnectivity` library. The trace level can be set to HIGH (3), MEDIUM (2), LOW (1), or NONE.

  ```
   + (void) setTraceLevel: (Level) value_in
  ```
- The APIs to enable tracing are available in the `ODPRequest` class.

- Use the `startTrace` API to check if end to end tracing is enabled and thereafter start tracing the message.

  ```
  +(void)startTrace;
  ```
- Use the `stopTrace` API to disable end to end tracing and thereafter stop tracing the message.

  ```
  +(void)stopTrace;
  ```
- Use the `uploadTraceWithError` API to forward the generated business transaction XML to the server.

  ```
   +(void) uploadTraceWithError:(NSError**) error
  ```

Examples

- **Setting a Trace level**

```
@try{
    [SDMRequestBuilder setTraceLevel:HIGH];
    }
    @catch (NSException *exception) {
        NSLog(@"excption:  %@",[exception description]);
```

- **Starting a Trace**

```
@try{
    [ODPRequest startTrace];
    }
    @catch (NSException *exception) {
       NSLog(@"Exception: %@",[exception description]);
```

- **Stopping a Trace**

```
@try{
    [ODPRequest stopTrace];
    }
    @catch(NSException *exception){
     NSLog(@"Exception: %@",[exception description]);
```

- **Uploading a Business Transaction XML**

```
@try{
     NSError* error = nil;
    [ODPRequest uploadTraceWithError:&error];
    }
```

Using Tracing APIs in Standalone SUPSupportability Library

You can perform end to end tracing using standalone SUPSupportability library.

SUPSupportability library provides two functionalities:

- generation of SAP Passport and Correlation ID for HTTP request and response messages.
- handling the generation of structured E2E Business Transaction XML .

The basic structure of the data is organized into:

- **SUPE2ETraceTransaction** - serves for the single client side trace
- **SUPE2ETraceStep** - denotes single user interaction roundtrip on the client side
- **SUPE2ETraceRequest** - denotes single interaction within an **SUPE2ETraceStep** resulting in an external interaction with backend server

To use the SUPSupportability library, import `SUPE2ETraceController.h` in the header file.

Usage

Follow the instructions to enable end to end tracing using the standalone SUPSupportability library. Once you have the traces loaded into a Business Transaction XML, you can upload to SAP Solution Manager.

1. Once tracing is enabled, create an instance of SAP Passport.

```
SAP_ExtendedPassport* passport = [SAP_ExtendedPassport
createPassport];
```

2. Create an instance of TraceController and TraceTransaction.

```
SUPE2ETraceController* traceController = [SUPE2ETraceController
sharedController];
SUPE2ETraceTransaction* traceTransaction = [traceController
createTrasaction];
```

3. Initialize TraceStep and TraceRequest objects before forwarding any new requests.

```
SUPE2ETraceStep * traceStep = [traceController createStep];
SUPE2ETraceRequest * traceRequest = [traceController
createRequest];
```

4. Set the tracing level.

```
[traceController setTraceLevel:1];
```

You can set any of the following trace levels.

Trace Level	Description
0	Specific to trace analysis on the client. No traces are triggered on the server.
1	Corresponds to response time- distribution analysis. This helps to analyse the time taken on each server component.
2	Corresponds to performance analysis. Performance traces are triggered on the server side. Example: Introscope Transaction Trace, ABAP Trace, SQL Traces and so on.

Trace Level	Description
3	Corresponds to functional analysis.

5. Add the SAP Passport and Correlation ID in the HTTP request header and send it.

```
[m_HTTPRequest addRequestHeader:@"SAP-PASSPORT" value:[theRequest
PassportHttpHeader]];
[m_HTTPRequest addRequestHeader:@"X-CorrelationID" value:
[theRequest CorrelationIdHttpHeader]];
 [theRequest markSending]
```

6. Record all updates made to requests in the Business Transaction XML.

```
[traceController updateRequestElementsInRequest:traceRequest
withDictionary:Dictionary];
```

7. Mark the data after receiving the first and last bit of data.

```
[theRequest markReceiving]
[theRequest markReceived]
```

8. Update the request with HTTP response header information. End the trace request and step objects.

```
[traceStep endStep];
[traceRequest endRequest];
```

9. Generate the Business Transaction XML and upload it to the server.

```
[traceController getXML];
[traceController sendXML:<SAP Solution Manager>]
```

Supportability API Reference for iOS

Use the Supportability API Reference as the primary reference for all standalone Supportability API listings.

SAP_ExtendedPassport class

comment the following lines out if you do not have the required exception classes

Syntax

```
public class  SAP_ExtendedPassport
```

Remarks

```
Class leveraging SAPPassport functionality

SAP_ExtendedPassport* passport = [SAP_ExtendedPassport
createPassport];





              [m_HTTPRequest addRequestHeader:@"SAP-PASSPORT" value:
[passport passportHttpHeader]];
[m_HTTPRequest addRequestHeader:@"X-CorrelationID" value:[passport
corrrelationIdHttpHeader]];
```

correlationIdHttpHeader() method
Retrieves the @“X-CorrelationID“ header value.

Syntax
```
public virtual NSString * correlationIdHttpHeader ()
```

createPassport() method
Factory method Returns an autoreleased SAPExtendedPassport instance initialized with defaults (medium trace level and auto generated UID).

Syntax
```
public static virtual SAP_ExtendedPassport * createPassport
()
```

*createPassportWithRootId:andTraceLevel:(NSData *, TraceLevel) method*
Factory method Returns an autoreleased SAPExtendedPassport instance.

Syntax
```
public static virtual SAP_ExtendedPassport *
createPassportWithRootId:andTraceLevel: (NSData * rootId_in,
TraceLevel traceLevel_in)
```

getuuidAsData() method
Factory method Returns an autoreleased UUID instance.

Syntax
```
public static virtual NSData * getuuidAsData ()
```

getuuidAsString() method
Factory method Returns an autoreleased UUID instance as HEXASCII-String.

Syntax
```
public static virtual NSString * getuuidAsString ()
```

*initWithRootId:andTraceLevel:(NSData *, TraceLevel) method*
Initializes a Passport object.

Syntax
```
public virtual id initWithRootId:andTraceLevel: (NSData *
RootId, TraceLevel traceLevel_in)
```

Usage

transportId and terminalId will be set to nil

*initWithRootId:andTraceLevel:andTransportId:andTerminalId:(NSData *, TraceLevel, NSData *, NSData *) method*

Initializes a Passport object.

Syntax

public virtual id initWithRootId:andTraceLevel:andTransportId:andTerminalId:(NSData * **rootId_in**, TraceLevel **traceLevel_in**, NSData * **transportId_in**, NSData * **terminalId_in**)

passportHttpHeader() method

Retrieves the @“SAP-PASSPORT“ header value.

Syntax

```
public virtual NSString * passportHttpHeader ()
```

SUPE2ETraceController class

Class leveraging BusinessTransaction.xml and SAPPassport functionality.

Syntax

```
public class SUPE2ETraceController
```

Remarks

```


SUPE2ETraceController* TraceHandler = [SUPE2ETraceController
sharedController];
SUPE2ETraceTransaction* Trace = [TraceHandler createTransaction];
SUPE2ETraceStep* theStep = [Trace createStep];
SUPE2ETraceRequest* theRequest = [theStep createRequest];










[m_HTTPRequest addRequestHeader:@“SAP-PASSPORT“ value:[theRequest
PassportHttpHeader]];
[m_HTTPRequest addRequestHeader:@“X-CorrelationID“ value:[theRequest
CorrelationIdHttpHeader]];
```

createRequest() method

Syntax

```
public virtual SUPE2ETraceRequest * createRequest ()
```

createStep() method

starts a new step with an associated Requestlist container.

Syntax

```
public virtual SUPE2ETraceStep * createStep ()
```

createTransaction() method

returns a TraceController singleton reference.

Syntax

```
public virtual SUPE2ETraceTransaction * createTransaction ()
```

*createTransactionWithName:withClientHost:(NSString *, NSString *) method*

returns a TraceController singleton reference.

Syntax

```
public virtual SUPE2ETraceTransaction *
createTransactionWithName:withClientHost: (NSString *
transactionName, NSString * clientHost)
```

*endRequest:(SUPE2ETraceRequest *) method*

Syntax

```
public virtual Boolean endRequest: (SUPE2ETraceRequest *
theRequest)
```

*endStep:(SUPE2ETraceStep *) method*

ends step with an associated Requestlist container.

Syntax

```
public virtual Boolean endStep: (SUPE2ETraceStep * theStep)
```

endTransaction() method

Syntax

```
public virtual Boolean endTransaction ()
```

*getStepAndRequestByPassportHeader:(NSString *) method*
returns a dictionary containing References to Step and Request Object for a given PassportHttpHeader.

Syntax

```
public virtual NSDictionary *
getStepAndRequestByPassportHeader: (NSString * Passport_in)
```

Usage

If developers store the PassportHttpHeader with their own objects they can retrieve the relevant Step and request object references later with PassportHeader as lookup parameter.

*getStepAndRequestForKey:withSearchString:(NSString *, NSString *) method*
returns a dictionary containing References to Step and Request Object for a given Key/Value pair within an SUPE2ETraceRequest object.

Syntax

```
public virtual NSDictionary *
getStepAndRequestForKey:withSearchString: (NSString * Key_in,
NSString * Value_in)
```

Parameters

- **Key_in –** Key field in SUPE2ETraceRequest
- **Value_in –** Searchvalue for Key field in SUPE2ETraceRequest

*getStepAndRequestForSearchValue:(NSString *) method*
returns a dictionary containing references to Step and SUPE2ETraceRequest object for a given searchvalue within an SUPE2ETraceRequest object.

Syntax

```
public virtual NSDictionary *
getStepAndRequestForSearchValue: (NSString * Value_in)
```

Parameters

- **Value_in –** Searchvalue for SearchKey field in SUPE2ETraceRequest

getXML() method
returns generated Business Transaction XML.

Syntax
```
public virtual NSString * getXML ()
```

Returns
String XML Usage [[SUPE2ETraceController sharedController] getXML];

*sendXML:(NSURL *) method*
Uploads the generated BTX to provided server URL and returns the response status code.

Syntax
```
public virtual NSUInteger sendXML: (NSURL * toBaseURL)
```

Parameters

- **baseUrl –** server Url Usage [[SUPE2ETraceController sharedController] sendXML:solmanURL];

Returns
Response status code as Integer

*setSearchKeyForRequest:WithValue:(SUPE2ETraceRequest *, NSString *) method*
SUPE2ETraceRequest defines a custom searchkey field for use by developers to be set with a application specific value for later retrieval of the SUPE2ETraceRequest and SUPE2ETraceStep object reference.

Syntax
```
public virtual Boolean setSearchKeyForRequest:WithValue:
(SUPE2ETraceRequest * theRequest_in, NSString * Value_in)
```

Usage

Code Example: BusTransTests.m accessing SUPE2ETraceController ... BusTransTests.m setting a searchvalue

setTraceLevel:(NSUInteger) method
sets the E2E Tracelevel with the TraceController

Syntax
```
public virtual void setTraceLevel: (NSUInteger trcLvl)
```

Parameters

- **trcLvl –** integer representing the tracelevel (LOW=1,MEDIUM=2,HIGH=3)

sharedController() method
returns a TraceController singleton reference.

Syntax
```
public static virtual SUPE2ETraceController *
sharedController()
```

*updateRequestElementsInRequest:withDictionary:(SUPE2ETraceRequest *, NSDictionary *) method*
Updates the data for each request and step to form transaction step xml and append it to the business transaction xml.

Syntax
```
public virtual Boolean
updateRequestElementsInRequest:withDictionary:
(SUPE2ETraceRequest * theRequest, NSDictionary * withDictionary)
```

Parameters

- **request –** current tracerequest
- **dictionary –** Contains the references to step and traceRequest object with passportheader and co-relationID

*updateRequestElementWithValue:::(SUPE2ETraceRequest *, NSString *, NSString *) method*
Updates the data for each request and step to form transaction step xml and append it to the business transaction xml.

Syntax
```
public virtual Boolean updateRequestElementWithValue:::
(SUPE2ETraceRequest * theRequest, NSString * Value, NSString *
forKey)
```

Parameters

- **request –** current tracerequest
- **Value –** step and tracerequest object value
- **key –** step and tracerequest object key

SUPE2ETraceRequest class

Syntax

```
public class  SUPE2ETraceRequest
```

CorrelationIdHttpHeader() method

Retrieves the @“X-CorrelationID“ header value.

Syntax

```
public virtual NSString *  CorrelationIdHttpHeader ()
```

Usage

```
Usage

[m_HTTPRequest addRequestHeader:@“X-CorrelationID“ value:[theRequest
CorrelationIdHttpHeader]];
```

*createRequest:(NSString *) method*

Standard initialization.

Syntax

```
public static virtual SUPE2ETraceRequest *  createRequest:
(NSString * WithId)
```

Usage

The request created receives default attributes. Request timestamp is set to the time of initialization. SAP-PASSPORT and X-CorrelationID http headers are created.

endRequest() method

At the end when response is received.

Syntax

```
public virtual Boolean  endRequest ()
```

Returns

Boolean Usage SUPE2ETraceController* TraceHandler = [SUPE2ETraceController sharedController]; SUPE2ETraceTransaction* Trace = [TraceHandler createTransaction]; SUPE2ETraceStep* theStep = [Trace createStep]; SUPE2ETraceRequest* theRequest = [theStep createRequest]; [theRequest markReceived];

getXML() method

Syntax

```
public virtual NSString * getXML ()
```

markReceived() method

When all data from the response have been received call this.

Syntax

```
public virtual void markReceived ()
```

Returns

void Usage SUPE2ETraceController* TraceHandler = [SUPE2ETraceController sharedController]; SUPE2ETraceTransaction* Trace = [TraceHandler createTransaction]; SUPE2ETraceStep* theStep = [Trace createStep]; SUPE2ETraceRequest* theRequest = [theStep createRequest]; [theRequest markReceived];

markReceiving() method

When the first data is being received.

Syntax

```
public virtual void markReceiving ()
```

Returns

void Usage SUPE2ETraceController* TraceHandler = [SUPE2ETraceController sharedController]; SUPE2ETraceTransaction* Trace = [TraceHandler createTransaction]; SUPE2ETraceStep* theStep = [Trace createStep]; SUPE2ETraceRequest* theRequest = [theStep createRequest]; [theRequest markReceiving];

markSending() method

Should be called before the request is sent out.

Syntax

```
public virtual void markSending ()
```

Returns

void Usage SUPE2ETraceController* TraceHandler = [SUPE2ETraceController sharedController]; SUPE2ETraceTransaction* Trace = [TraceHandler createTransaction]; SUPE2ETraceStep* theStep = [Trace createStep]; SUPE2ETraceRequest* theRequest = [theStep createRequest]; [theRequest markSending];

markSent() method
Should be called as soon as the data is being sent.

Syntax
```
public virtual void markSent ()
```

Returns
void Usage SUPE2ETraceController* TraceHandler = [SUPE2ETraceController sharedController]; SUPE2ETraceTransaction* Trace = [TraceHandler createTransaction]; SUPE2ETraceStep* theStep = [Trace createStep]; SUPE2ETraceRequest* theRequest = [theStep createRequest]; [theRequest markSent];

PassportHttpHeader() method
Retrieves the @“SAP-PASSPORT“ header value.

Syntax
```
public virtual NSString * PassportHttpHeader ()
```

Usage
```
Usage

[m_HTTPRequest addRequestHeader:@"SAP-PASSPORT" value:[theRequest
PassportHttpHeader]];
```

*updateRequestElementsWithDictionary:(NSDictionary *) method*

Syntax
```
public virtual Boolean updateRequestElementsWithDictionary:
(NSDictionary * withDictionary)
```

*updateRequestWithValue:withValue:(NSString *, NSString *) method*

Syntax
```
public virtual Boolean updateRequestWithValue:withValue:
(NSString * , NSString * forKey)
```

SUPE2ETraceSettings class

Syntax
```
public class SUPE2ETraceSettings
```

incrementRequest() method

Syntax

```
public virtual void incrementRequest ()
```

incrementStep() method

Syntax

```
public virtual void incrementStep ()
```

resetRequest() method

Syntax

```
public virtual void resetRequest ()
```

resetStep() method

Syntax

```
public virtual void resetStep ()
```

setTraceFlags:Byte0:(Byte, Byte) method

Syntax

```
public virtual void setTraceFlags:Byte0: (Byte , Byte Byte1)
```

sharedManager() method

Syntax

```
public static virtual SUPE2ETraceSettings * sharedManager ()
```

TraceFlags() method

Syntax

```
public virtual NSData * TraceFlags ()
```

TraceFlagsAsString() method

Syntax

```
public virtual NSString * TraceFlagsAsString ()
```

SUPE2ETraceStep class

Syntax
```
public class  SUPE2ETraceStep
```

addUIAction() method

Syntax
```
public virtual void  addUIAction ()
```

createStep() method

Syntax
```
public static virtual SUPE2ETraceStep *  createStep ()
```

endStep() method

Syntax
```
public virtual Boolean  endStep ()
```

getXML() method

Syntax
```
public virtual NSString *  getXML ()
```

SUPE2ETraceTransaction class

Syntax
```
public class  SUPE2ETraceTransaction
```

createTransaction() method

Syntax
```
public static virtual SUPE2ETraceTransaction *
createTransaction ()
```

*createTransactionWithName:withClientHost:(NSString *, NSString *) method*

Syntax
```
public static virtual SUPE2ETraceTransaction *
createTransactionWithName:withClientHost: (NSString *
transactionName, NSString * clientHost)
```

endTransaction() method

Syntax

```
public virtual Boolean  endTransaction ()
```

getXML() method

Syntax

```
public virtual NSString *  getXML ()
```

Analyzing Performance Data Points

(Optional) To analyze the performance of the client, measurement points are available at different stages in a request-response cycle. These points are used to provide logs that help in assessing the processing time across various components in the SAP Mobile Platform environment.

Note: The response time will be impacted after importing performance library. If you set the custom settings in SAP Control Center as true for logs, then there will be a degradation in the response time.

Data Points for the Client

The table below provides the list of data points and the log readings across which the performance can be measured.

Log Reading	Application	Proxy Client	Messaging Client	Network (includes reverse proxy/ relay server)	SAP Mobile Platform Server	Network	Enterprise Information System (EIS)
E2E: RR		X	X	X	X	X	X
ODP:RR			X	X	X	X	X
IMO:RR			X	X	X	X	X
Net-work:RR				X	X	X	X

Performance Readings

The above log readings determine the time elapsed across various stages of a request response (RR) cycle.

- E2E:RR - Corresponds to the performance reading when the request is forwarded from the proxy client and the response reaches the proxy client.
- ODP:RR - Corresponds to the performance reading when the request reaches the Messaging Client and the response reaches the Messaging Client.
- IMO:RR - Corresponds to the performance reading when the request is forwarded from the Messaging Client and the response reaches the Messaging Client.
- Network:RR - Corresponds to the performance reading when the request is forwarded from the network and the response reaches the network.

The above log readings determine the time elapsed across various stages of a request response (RR) cycle. Use these readings to determine processing time across the following components:

- Time taken at the messaging client: **ODP:RR - IMO:RR**
- Time taken at the proxy client: **E2E:RR - ODP:RR**

Development Task Flow Using REST SDK (HTTP Channel)

Describes the overall development task flow for ODP applications using REST SDK. An application consists of building blocks which the developer uses to start the application and perform functions needed for the application.

1. *Creating and Initializing the Client Connection*

   Create and initialize SMPClientConnection.
2. *Enabling Network Edge for HTTP*

   For network edge scenario, all the requests through reverse proxy (example, Microsoft ARR) should be protected by SiteMinder Policy Server through some pattern. Network edge works only in asynchronous registration.
3. *Enabling Single and Mutual SSL Authentication*

   Secure sockets layer (SSL) is a protocol that governs certificate authentication. It exchanges the certificate information. SSL also encrypts all information that flows between a client and a server. Mutual SSL authentication or certificate based mutual authentication refers to two parties authenticating each other through verifying the provided digital certificate, so that both parties are assured of the others' identity.
4. *Registering the User*

   Registers the user to the SAP Mobile Platform on-premise or cloud.
5. *Enabling CAPTCHA*

   Enable CAPTCHA during registration.
6. *Exchanging Settings between Client and Server*

   Settings exchange between the client and the SAP Mobile Server using REST SDK.

7. *Setting Password Policy Using DataVault*

   (Optional) Use the getPasswordPolicy method to set password policy using DataVault.
8. *Subscribing for Apple Push Notifications*

   (Optional) Subscribe for Apple Push Notifications (APNS).
9. *End to End Tracing*

   (Optional) End to end tracing enables an application developer and end user to trace a request that is sent from the client to the back-end. This spans the entire landscape where you can derive a correlation of traces at the client, server and back-end.
10. *Sending Data Request to the Backend*

    Send a data request to the back-end through the SAP Mobile Server asynchronously.
11. *Retrieving the Response from Backend*

    Retrieve the response from the backend to the device.
12. *Deleting a User*

    Delete a registered user.
13. *REST SDK API Usage*

See also

- *Development Task Flow for iOS Applications Using OData SDK (Messaging Channel)* on page 12
- *Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)* on page 77

Creating and Initializing the Client Connection

Create and initialize SMPClientConnection.

The `SMPClientConnection` class declares the programmatic interface for an object that manages connection settings required for registering the user and fetching application settings from the server. An `SMPClientConnection` object has to be initialized and connection properties has to be set on this object before performing any type of user onboarding or before fetching any application settings from the server.

Syntax

The following code illustrates how to initialize an application:

```
+(SMPClientConnection*)initializeWithAppID:(NSString*)applicationID
domain:(NSString*)domain secConfiguration:(NSString*)secConfig;
```

Examples

- **Example for initializing an application –** Returns an instance of `SMPClientConnection` class.

```
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:@"<application_id>" domain:@"default"
secConfiguration:@"SSO"];
```

Enabling Network Edge for HTTP

For network edge scenario, all the requests through reverse proxy (example, Microsoft ARR) should be protected by SiteMinder Policy Server through some pattern. Network edge works only in asynchronous registration.

The application should set the selector for receiving authentication challenge in Siteminder scenario.

Examples

- **Siteminder call back code for asynchronous registration:**

```
[SMPUserManager setDelegate:self];
 NSError *error = nil;

    l_clientconn = [SMPClientConnection
initializeWithAppID:@"<application_id>"domain:@"default"secConfig
uration:@"SMNetworkEdge"];

    [l_clientconn setConnectionProfileWithUrl:@"http://
<host:port>/<relayserver_url template>/<Relayserver_farmID>"];

    [SMPUserManager
setAuthChallengeSelector:@selector(authenticationNeededForRequest
:)];
    [SMPUserManager SetDelegate:self]

    userManager = [SMPUserManager
initializeWithConnection:l_clientconn];


    [userManager  registerUser:nilpassword:nilerror:&error
isSyncFlag:NO];


-(void)authenticationNeededForRequest:(SDMHttpRequest*)request
{
    [request setUsername:@"<GatewayUsername>"];
    [request setPassword:@"<GatewayPassword>"];
    [request retryUsingSuppliedCredentials];
}
```

Enabling Single and Mutual SSL Authentication

Secure sockets layer (SSL) is a protocol that governs certificate authentication. It exchanges the certificate information. SSL also encrypts all information that flows between a client and a server. Mutual SSL authentication or certificate based mutual authentication refers to two parties authenticating each other through verifying the provided digital certificate, so that both parties are assured of the others' identity.

In single SSL connection:

- The CA certificate should be installed on the device trust store which the application takes while connecting to the HTTPS URL. The SSL connection is established.
- Registering the user through single SSL, use the code:

```
[SMPUserManager setDelegate:self];
l_clientconn = [SMPClientConnection
initializeWithAppID:<applicationId> domain:<domain>
secConfiguration:<securityconfiguration>];
      userManager = [SMPUserManager
initializeWithConnection:l_clientconn];
//During onboarding with single SSL, add fully qualified domain
name of the SAP Mobile Platform server. For example,
vwxxx.dhcp.wdf.sap.corp.
[l_clientconn setConnectionProfileWithHost:<host> port:<port>
farm:<farm> relayServerUrlTemplate:<relayserverURLtemplate>
enableHTTP:NO];

[userManager registerUser:<backendusername>
password:<backendpassword> error:&error isSyncFlag:NO];
```

After initializing the `SMPClientConnection` class, use the `setClientIdentityCertificate` method to set the client certificate.

Examples

- **Mutual SSL code:**

```
[SMPUserManagersetDelegate:self];
l_clientconn =
[SMPClientConnectioninitializeWithAppID:applicationIddomain:@"def
ault"secConfiguration:odp_cert_sc];

    userManager =
[SMPUserManagerinitializeWithConnection:l_clientconn];

    NSError *error = nil;

    SecIdentityRef identityApp = nil;

    NSString *path = [[NSBundlemainBundle]
pathForResource:@"Certificate"ofType:@"p12"];        /// This is
your client certificate
    NSLog(@"path is %@",path);
    NSData *PKCS12Data = [[NSData
alloc]initWithContentsOfFile:path];

    CFDataRef inPKCS12Data = (__bridge CFDataRef)PKCS12Data;
    CFStringRef password = CFSTR("mobile");
    /// Password of p12 certificate
    const void *key[] = { kSecImportExportPassphrase };//
kSecImportExportPassphrase };
    const void *values[] = { password };
    CFDictionaryRef options = CFDictionaryCreate(NULL, key,
values, 1, NULL, NULL);
    CFArrayRef items = CFArrayCreate(NULL, 0, 0, NULL);
    OSStatus securityError = SecPKCS12Import(inPKCS12Data,
options, &items);
    CFRelease(options);
    CFRelease(password);
    if (securityError == errSecSuccess)
    {
        NSLog(@"Success opening p12 certificate. Items: %ld",
CFArrayGetCount(items));
        CFDictionaryRef identityDict =
CFArrayGetValueAtIndex(items, 0);
        identityApp =
(SecIdentityRef)CFDictionaryGetValue(identityDict,
kSecImportItemIdentity);
    } else
    {
        NSLog(@"Error opening Certificate.");
    }


    [l_clientconnsetClientIdentityCertificate:identityApp];

     //During onboarding with single SSL, add fully qualified
domain name of the SAP Mobile Platform server. For example,
vwxxx.dhcp.wdf.sap.corp.
    //If relay server is used, <host> is relay server host, <port>
is relay server port, and <farm> is relay server farm ID, and
provide the corresponding relay server URL template. If fully
qualified domain name is used, the <host> is FQDN of SAP Mobile
Platform server, <port> is 8002 for mutual SSL, and 8001 for
single SSL, <farm> is 0, and <relayserverURLtemplate> is nil.
    [l_clientconn setConnectionProfileWithHost:<host> port:<port>
farm:<farm> relayServerUrlTemplate:<relayserverURLtemplate>
enableHTTP:NO];

      [userManager registerUser:<backendusername>
password:<backendpassword> error:&error isSyncFlag:NO];
```

Registering the User

Registers the user to the SAP Mobile Platform on-premise or cloud.

You need to send username and password as part of request manager during `ClientConnection` initialization. **isSynchronous** specifies whether this is a synchronous or an asynchronous registration. If asynchronous, then you have to implement and register `IODPUserRegistrationListener` to get a call back.

Syntax

```
 -(BOOL)registerUser:(NSString*)userName password:
(NSString*)loginPassword
 error:(NSError**)error isSyncFlag:(BOOL)isSynchronous;
```

Examples

- **Example for registering the user using `setConnectionProfileWithUrl`**

```
{
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:@"<application_id>" domain:@"default"
secConfiguration:@"<Security_configuration>"];
[clientConn setConnectionProfileWithUrl:@"http://
<server_host>:<server_http_port>"];
NSError* error = nil;
[SMPUserManager setDelegate:self];
[SMPUserManager setCaptchaChallengeDelegate:self];
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];
[userManager registerUser:@"GW_Username" password:@"GW_password"
error:&error isSyncFlag:NO];
//…
//…
}
-(void)userRegistrationSuccessful
{

}
-(void)userRegistrationFailed:(NSError*)error
{
// Error handling using the error object
NSLog(“%@”,error);
}
-(NSString*)didReceiveCaptchaChallenge:
(NSString*)base64ImageString
{
   NSError* error = nil;
   [userManager registerUser:@"GW_Username"
password:@"GW_password" captchaText:@"captchText" error:&error
isSyncFlag:NO])
   return nil;
}
```

- **Example for registering the user using `setConnectionProfileWithHost`**

```
{
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:@"<application_id" domain:@"default"
secConfiguration:@"<Security_configuration>"];
[clientConn setConnectionProfileWithHost:@"<server_host/
relayserver_host>"port:@"<http_port/relayserver_port>"
farm:<nil/relaysrever_farmid> relayServerUrlTemplate:<nil/
relayserverURLtemplate>  enableHTTP:YES];

NSError* error = nil;
[SMPUserManager setDelegate:self];
[SMPUserManager setCaptchaChallengeDelegate:self];
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];
[userManager registerUser:@"GWUsername" password:@"GWPassword"
error:&error isSyncFlag:NO];
//…
//…
}
-(void)userRegistrationSuccessful
{

}
-(void)userRegistrationFailed:(NSError*)error
{
// Error handling using the error object
NSLog(“%@”,error);
}
//For added security, use CAPTCHA when registering in a Cloud
environment
-(NSString*)didReceiveCaptchaChallenge:
(NSString*)base64ImageString
{
   NSError* error = nil;
   [userManager registerUser:@"GW_Username"
password:@"GW_password" captchaText:@"captchText" error:&error
isSyncFlag:NO])
   return nil;
}
```

Enabling CAPTCHA

Enable CAPTCHA during registration.

Syntax

Use `SMPCaptchaChallengeDelegate` to enable CAPTCHA challenge for registration.

```
+(void)setCaptchaChallengeDelegate:
(id<SMPCaptchaChallengeDelegate>)delegate;
```

Examples

- **Implement CAPTCHA challenge during registration –** The following example illustrates how to implement the `SMPCaptchaChallengeDelegate` and register this delegate to get notified of captcha challenges from the server.

```
        {
        SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:@"com.sap.NewFlight" domain:@"default"
        secConfiguration:@"SSO"];
        [clientConn setConnectionProfileWithUrl:@"https://
mobilesmp.hana.ondemand.com"];
        [SMPUserManager setCaptchaChallengeDelegate:self];
        SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];
        //…
        //…   }
       -(NSString*)didReceiveCaptchaChallenge:
(NSString*)base64ImageString
        {
         return captcha string ;
        }
```

Exchanging Settings between Client and Server

Settings exchange between the client and the SAP Mobile Server using REST SDK.

Update application settings.

Syntax

Get the application setting configured for the user.

```
-(NSDictionary*)getConfigPropertyMapWithError:(NSError**)error;
```

Fetch the configuration property for the user using:

```
-(NSString*)getConfigProperty:(NSString*)configKey error:
(NSError**)error;
```

Or

```
NSString *deviceToken=[l_appsettingsObject
getConfigProperty:@"d:ApnsDeviceToken"
      error:&error];
```

Update the application settings configured for the user.

```
-(BOOL)setConfigProperty:(NSDictionary*)settingsList error:
(NSError**)error;
```

Examples

- **Example to set a configuration property for the user**

```
NSError* error = nil;
NSDictionary *props = [NSMutableDictionary dictionary];
[props setValue:@"2134hjbuh243g32234bhjb786"
forKey:@"d:ApnsDeviceToken"];
[appSettings setConfigProperty:props error:&error];
if (error)
 {
    NSLog(@"ERROR:  %@",error);
 }
NSDictionary* props = [appSettings
getConfigPropertyMapWithError:nil];
```

See also

- *Setting Password Policy Using DataVault* on page 50

Downloading Customization Resource Bundles

(Optional) Download customization resource bundles as defined in SAP Mobile Server

Syntax

```
-(NSData*)getCustomizationResourceBundleWithCustomizationResource:
(NSString*)
        customizationResource error:(NSError**)error;
```

Examples

- **Example for downloading customization resource bundles**

  ```
  NSError* _errorWhileFetchingID = nil;
  NSString *bundleId = [appSettings
  getConfigProperty:@"d:CustomizationBundleId"
  error:&_errorWhileFetchingID];
  if (!_errorWhileFetchingID)
  {
   NSError* _errorWhileFetchingBundle=nil;
   NSData* bundle = [appSettings
  getCustomizationResourceBundleWithCustomizationResource:bundleId
   error:&_errorWhileFetchingBundle];
   }
  //bundleId can be set to bundle name (fetches the specified
  bundle) or nil (fetches the default bundle selected)
  ```

Getting an Application Endpoint

Retrieve the application endpoint URL.

Returns the value of the application endpoint. It is used to perform request-response with the Gateway.

Syntax

```
-(NSString*)getApplicationEndpointWithError:(NSError**)error;
```

Examples

- **Example to get application endpoint URL**

  ```
  NSString* applicationEndpoint = [appSettings
        getApplicationEndpointWithError:nil];
  ```

Getting Push Endpoint

Retrieving push endpoint URL.

Returns the value of push endpoint. It is used to register for push notification.

Syntax

```
-(NSString*)getPushEndpointWithError:(NSError**)error;
```

Examples

- **Example to retrieve push endpoint URL**

  ```
  NSString* pushEndpoint = [appSettings
  getPushEndpointWithError:nil];
  ```

Setting Password Policy Using DataVault

(Optional) Use the `getPasswordPolicy` method to set password policy using DataVault.

Before calling the `getPasswordPolicy` method, set the `CapabilitiesPasswordPolicy` property to `true` using `setConfigProperty` method, so the user properties can be modified. By default, the value of `CapabilitiesPasswordPolicy` is set to `false`.

Syntax

```

NSDictionary *props = [NSMutableDictionary dictionary];
[props setValue:@"true" forKey:@"d:CapabilitiesPasswordPolicy"];
NSError* error = nil;
BOOL isSuccess = [appSettings setConfigProperty:props error:&error];
if (!error)
{
     // Further processing
}
NSError* errorFetchingPP = nil;
DataVaultPasswordPolicy *dvppStruct = [appSettings
getPasswordPolicy:& errorFetchingPP];
if (!errorFetchingPP)
{
 //  Further processing
}
```

See also

- *Exchanging Settings between Client and Server* on page 48

Subscribing for Apple Push Notifications

(Optional) Subscribe for Apple Push Notifications (APNS).

Enable the certificate for push.

Use the Apple Push Notification service to propagate information from the backend to the device. Enable APNS for iOS applications using the native iOS push services.

For more information on how to enable certificate for APNS, see Apple Web site: *http://developer.apple.com/xcode/*

Add the push certificate to Push Configurations tab in SCC.

For more information, see *Configuring Native Notifications in SAP Control Center for SAP Mobile Platform*.

Exchange device token using setConfigProperty method

Add these two methods into the `AppDelegate` class to fetch the device token at the client side:

```
-(void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
       *)deviceToken
 {
   NSString *devToken=[[[NSString alloc]initWithData:deviceToken
encoding:NSUTF8StringEncoding]autorelease];
   NSLog(@"Device Token:  %@",devToken);
 }
-(void)application:(UIApplication *)application
didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
 {
   NSLog(@"Error Occurred: %@",[error description]);
 }
```

After successful registration, set the device token received in `didRegisterForRemoteNotificationsWithDeviceToken` method using following code:

```
NSError* error = nil;
NSDictionary *props = [NSMutableDictionary dictionary];
[props setValue:deviceToken forKey:@"d:ApnsDeviceToken"];
[appSettings setConfigProperty:props error:&error];
   if (error) {
            NSLog(@"ERROR:  %@",error);
           }
```

End to End Tracing

(Optional) End to end tracing enables an application developer and end user to trace a request that is sent from the client to the back-end. This spans the entire landscape where you can derive a correlation of traces at the client, server and back-end.

These correlated traces help in performance analysis and are centrally monitored on SAP Solution Manager. These are displayed as reports where you can extract information on failure of delivering a request, time taken for a request to reach a component and so on.

On the client side, the client framework enables an application developer to switch on the trace for messages. The client traces the request at predefined points and all these transactions/ requests are recorded in a Business Transaction XML. Additionally, the client maintains a unique identifier in the HTTP header called the SAP Passport that is used to correlate traces across various components. This Business Transaction XML can later be uploaded to the SAP Solution Manager which is a central location to correlate all logging information.

The end to end tracing APIs are available in the `SMPClientConnection` class.

Note:

- For end to end logs to be added to the SAP Solution Manager, configure the Solution Manager URL in SAP Control Center. For more information, see *Configuring SAP Solution Manager URL* in *SAP Control Center for SAP Mobile Platform*.
- Ensure that the user is selected under domain logs of SAP Control Center to enable any end to end logs.

Syntax

- Use the `setTraceLevel` API to set the trace level in the SAP-PASSPORT and BTX . The trace level can be set to E2EHIGH (3), E2EMEDIUM (2), E2ELOW (1), or NONE.

  ```
  -(void)setTraceLevel:(E2ELevel)value_in;
  ```

  Note: If the trace level is not set using the API, trace level 3 is automatically chosen.
- Use the `startTrace` API to start end to end tracing on the client.

  ```
  -(void)startTrace;
  ```
- Use the `stopTrace` API to stop end to end tracing on the client.

  ```
  -(void)stopTrace;
  ```
- Use the `uploadTraceWithUserName` API to generate BTX on the client, and upload the BTX to Solution Manager via the SAP Mobile Platform server.

  ```
  -(void)uploadTraceWithUserName:(NSString*)username passWord:
  (NSString*)password
          error:(NSError**)error;
  ```

Examples

- **Enabling end to end tracing**

  ```
   SMPClientConnection* clientConn = [SMPClientConnection
  initializeWithAppID:@"com.sap.NewFlight" domain:@"default"
   secConfiguration:@"SSO"];
   [clientConn setConnectionProfileWithUrl:@"http://
  dewdf30185.wdf.sap.corp:8000"];

   //Successfully onboard the user

   [clientConn setTraceLevel:E2EHIGH];
   [clientConn startTrace];

   //Perform request response
  ```

```
  [clientConn stopTrace];
  NSError* _uploadError = nil;
  [clientConn uploadTraceWithUserName:@"supuser"
passWord:@"s3puser" error:&_uploadError];
   if(_uploadError)
 {
 NSLog(@"BTX UPLOAD ERROR: %@",_uploadError);
 }
```

Sending Data Request to the Backend

Send a data request to the back-end through the SAP Mobile Server asynchronously.

Initialize the `SMPAppSettings` class which takes client connection object as one of the parameter using `SMPAppSettings *appSettingsObj=[SMPAppSettings initializeWithConnection:clientConn userName:username password:password];`

```

/*Initialize the application settings class which takes client
connection object as one of the parameters */

   SMPAppSettings *appSettingsObj=[ SMPAppSettings
initializeWithConnection :clientConn  userName : @”supuser”
password : @”s3puser” ];

   /* Get the application endpoint using [appSettingsObj
getApplicationEndpointWithError:error]        */

    NSString * appEndPoint =[appSettingsObj
getApplicationEndpointWithError:nil];

   /*Initialize the SDMRequesting class object with the endpoint URL
after successful registration */
      id<SDMRequesting> request=nil;
      if([[self.m_reqTypeDic objectForKey:KEY_REQUEST_TYPE]
isEqualToString:SERVICE_DOC_REQUEST])
      {
       /* Get Service Document    */
         request = [SDMRequestBuilder requestWithURL:[NSURL
URLWithString:appEndPoint]];
       }
       else if([[self.m_reqTypeDic objectForKey:KEY_REQUEST_TYPE]
isEqualToString:METADATA_REQUEST])
       {
        /* Get metadata   */
        request=[SDMRequestBuilder requestWithURL:[NSURL
URLWithString:[NSString stringWithFormat:@"%@/
$metadata",m_AppEndPoint]]];
       }
        else if([[self.m_reqTypeDic objectForKey:KEY_REQUEST_TYPE]
isEqualToString:ODATA_REQUEST])
        {
         /* To get required flight collection. For example:
CarrierCollection */
```

```
         request=[SDMRequestBuilder requestWithURL:[NSURL
URLWithString:[NSString stringWithFormat:@"%@/CarrierCollection",
appEndPoint]]];
        }
         /*Set Username in SDMRequesting object*/
         [request setUsername: @"supuser"];

         /*Set Password in SDMRequesting object*/
         [request setPassword: @"s3puser"];

         /*Set the Delegate. This class must adhere to
SDMHttpRequestDelegate to get callback*/
         [request setDelegate:self];

         /*Call startAsynchronous API to request object to retreive
Data asynchronously in the call backs */
         [request startAsynchronous];
```

Retrieving the Response from Backend

Retrieve the response from the backend to the device.

Upon performing request using [`request startAsynchronous`], the response will arrive at either `- (void)requestFailed:(SDMHttpRequest*) request {}` or `- (void)requestFinished:(SDMHttpRequest*) request {}` depending on whether the transaction was a **Failure** or **Success** respectively.

Add the following in the `- (void)requestFinished(SDMHttpRequest*)` request method:

```
   /* The response parsing depends on the type of document queried
whether its service document or metadata or
    OData*/
    if([[self.m_reqTypeDic objectForKey:KEY_REQUEST_TYPE]
isEqualToString:SERVICE_DOC_REQUEST])
    {
     /*Initialize the SDMODataServiceDocumentParser class object for
parsing response service document
        data*/
        SDMODataServiceDocumentParser
        *svcDocParser=[[[SDMODataServiceDocumentParser
alloc]init]autorelease];
        /* Parses the service document XML and converts it to an
Objective-C service document object.*/
           [svcDocParser parse:[request responseData]];
          /* Service document instance can be accessed via the
"serviceDocument" property of the parser after
             parsing*/
             SDMODataServiceDocument
             *svcDoc=svcDocParser.serviceDocument;
         }

         else if([[self.m_reqTypeDic objectForKey:KEY_REQUEST_TYPE]
isEqualToString:METADATA_REQUEST])
```

```
         {
       /*Initialize the SDMODataMetaDocumentParser class object for
parsing response meta data*/
           SDMODataMetaDocumentParser
*metaDocParser=[[[SDMODataMetaDocumentParser
           alloc]initWithServiceDocument:svcDoc]autorelease];

          /*Parses and matches the schema with the service document
and its collections.*/

             [metaDocParser parse:[request responseData]];

             /* The parser creates the schema of the input service
document's collections and returns a collection by name */
               SDMODataCollection *carrierCollection=[svcDoc.schema
getCollectionByName:@"CarrierCollection"];

               }
               else if([[self.m_reqTypeDic
objectForKey:KEY_REQUEST_TYPE] isEqualToString:ODATA_REQUEST])
               {
              /* Initialize the SDMODataDataParser class object for
parsing any "inlined"entries or feed(s) when service
                  document is passed to the "initWithEntitySchema"
variant that accepts service document as
                input. If "inlined" feed(s) or entries should not be
returned pass nil as the service
                  document parameter or use SDMODataDataParser*
dataParser = [[SDMODataDataParser alloc] initWithEntitySchema:
entitySchema] */

               SDMODataDataParser *dataParser=[[[SDMODataDataParser
alloc]initWithEntitySchema: carrierCollection.entitySchema
andServiceDocument: svcDoc]autorelease];

               /*Parses a feed or an entry xml.*/
                 [dataParser parse:[request responseData]];


             /* The array of parsed entry/entries can be accessed via
the "entries" property of the parser after parsing.
                Array of SDMOdata Entries can be iterated and diplay
the requisite data in tableview */

                NSMutableArray * carriersList=[l_dataParser.entries
retain];
         }
```

Deleting a User

Delete a registered user.

The deleteUser method is used to delete a registered (onboarded) user.

Syntax

```
-(BOOL)deleteUser:(NSString*)userName password:(NSString*)passWord
        error:(NSError**)error;
```

Examples

- **Delete a user –** To delete an already registered user:

  ```
  SMPClientConnection* clientConn =
  [SMPClientConnectioninitializeWithAppID:@"<application_id>"
  domain:@"default"secConfiguration:@"<security_configuration"];
  [clientConn setConnectionProfileWithHost:@"<server_host>"
  port:@"<server_port>"
  farm:nil relayServerUrlTemplate:nil enableHTTP:YES];
  NSError* error = nil;
  SMPUserManager* userManager = [SMPUserManager
  initializeWithConnection:clientConn];
  [userManager registerUser:@"<GWUsername>"
  password:@"<GWPassword>" error:&error isSyncFlag:NO];
  //..
  //..
  //..
  if([userManager deleteUser:@"supuser" password:@"s3puser"
  error:&errors])
  {
      NSLog(@"supuser deleted...");
  }
  ```

REST SDK API Usage

Rest Client API Reference for iOS

Use the Rest Client API reference as the primary reference for all API listings and error code information.

Refer to the Rest Client SDK API reference for each available package.

SMPAppSettings class

This class implements instance methods to fetch settings like end-points for application data access, push endpoint, customizationbundleID and password policy.

Syntax

```
public class  SMPAppSettings
```

*getApplicationEndpointWithError:(NSError **) method*

Retrieves the application end-point with which the application can access business data.

Syntax

```
public virtual NSString *  getApplicationEndpointWithError:
(NSError ** error)
```

Parameters

- **error –** Double pointer to the error object if the method results in an error.

Returns

Returns the end-point as a string object (URL/URN)

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
NSError *error = nil;
NSString *appEndPoint = [appSettings
getApplicationEndpointWithError:&error];
if (!error)
{
//Continue processing
}
```

Usage

Returns the end-point as a string object (URL/URN)

*getConfigProperty:error:(NSString *, NSError **) method*

Retrieves the Configuration Property by providing key.

Syntax

`public virtual NSString * getConfigProperty:error: (NSString *` **configKey**`, NSError **` **error**`)`

Parameters

- **configKey –** Key for the desired configuration property
- **error –** Double pointer to the error object if the method results in an error.

Returns

Returns the Configuration Property as a string object.

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
NSError* error = nil;
```

```
NSString* configProp = [appSettings getConfigProperty:configKey
error:&error];
if (!error)
{
// Further processing
}
```

Usage

Returns the Configuration Property as a string object.

*getConfigPropertyMapWithError:(NSError **) method*
Returns all the settings in Key-Value pair.

Syntax

```
public virtual NSDictionary *  getConfigPropertyMapWithError:
(NSError ** error)
```

Parameters

- **error –** Double pointer to the error object if the method results in an error.

Returns

Returns all the settings as NSDictionary object.

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
NSError* error = nil;
NSDictionary* configProp = [appSettings
getConfigPropertyMapWithError:&error];
if (!error)
{
// Further processing
}
```

Usage

Returns all the settings as NSDictionary object.

*getCustomizationResourceBundleWithCustomizationResource:error:(NSString *, NSError **) method*

Returns or downloads Customization Resources as binary data.

Syntax

`public virtual NSData * getCustomizationResourceBundleWithCustomizationResource:error: (NSString *` **customizationResource**`, NSError **` **error**`)`

Parameters

- **customizationResource : –** Customization Resource name.
- **error –** Double pointer to the error object if the function results in an error. Returns the Customization Resource(zip or jar file) as binary data.

Returns

Returns the Customization Resource as binary data.

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
NSError* error = nil;
NSData* customResourceBundleData = [appSettings
getCustomizationResourceBundleWithCustomizationResource:customiza
tionResource error:&error];
if (!error)
{
// Further processing
}
```

Usage

Returns the Customization Resource as binary data.

*getPasswordPolicy:(NSError **) method*

Retrieves the password policy set by the administrator in SCC for the application.

Syntax

`public virtual DataVaultPasswordPolicy * getPasswordPolicy: (NSError **` **error**`)`

Parameters

- **error –** Double pointer to the error object if the method results in an error.

Returns

Returns the structure containing the password policy.

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
NSError* error = nil;
DataVaultPasswordPolicy *dvppStruct = [appSettings
getPasswordPolicy:&error];
if (!error)
{
// Further processing
}
```

Usage

Returns the structure containing the password policy.

*getPushEndpointWithError:(NSError **) method*

Retrieves the push end-point which the enterprise back-end can use to push data to the ODP client via the server.

Syntax

```
public virtual NSString *  getPushEndpointWithError: (NSError
** error)
```

Parameters

- **error –** Double pointer to the error object if the method results in an error.

Returns

Returns the end-point as a string object (URL/URN)

Examples

- **Example**

```
-(void)userRegistrationSuccessful:(SMPUserManager*)userManager
{
NSLog("userRegistrationSuccessful...");
NSString* appConID= [userManager applicationConnectionID];
```

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
NSError *error=nil;
NSString *pushEndpoint = [appSettings
getPushEndpointWithError:&error];
if(!error){
NSLog("pushEndpoint : %",pushEndpoint);
}
```

Usage

This end-point will be registered with the enterprise back-end during the process of subscription.

Returns the end-point as a string object (URL/URN)

*initializeWithConnection:userName:password:(SMPClientConnection *, NSString *, NSString *) method*
Gets the singleton instance/object of the AppSettings class.

Syntax

`public static virtual SMPAppSettings * initializeWithConnection:userName:password: ( SMPClientConnection *` **clientConn**`, NSString *` **userName**`, NSString *` **password**`)`

Parameters

- **clientConn –** Object of SMPClientConnection class
- **userName –** User Name that will be authorised by a predefined security provider
- **password –** Password for that user

Returns

Returns the instance of the class SMPAppSettings

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
```

Usage

Returns the instance of the class SMPAppSettings

*releaseStoredCredentialsWithError:(NSError **) method*

Releases the username and password provided while initializing app settings class.

Syntax

`public virtual void releaseStoredCredentialsWithError: (NSError ** `**`error`**`)`

Parameters

- **error –** Double pointer to the error object if the method results in an error.

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
NSError* error = nil;
[appSettings releaseStoredCredentialsWithError:&error];
if (!error)
{
// Further processing
}
```

*setConfigProperty:error:(NSDictionary *, NSError **) method*

Updates the config property by providing list of settings as a NSDictionary Object.

Syntax

`public virtual BOOL setConfigProperty:error: (NSDictionary * `**`settingsList`**`, NSError ** `**`error`**`)`

Parameters

- **settingsList –** List of settings as key-value
- **error –** Double pointer to the error object if the method results in an error.

Returns

Returns BOOL whether successfully updated or not.

Examples

- **Example**

```
SMPAppSettings* appSettings = [SMPAppSettings
initializeWithConnection:clientConn userName:"username"
password:@"password"];
```

```
NSDictionary *props = [NSMutableDictionary dictionary];
[props setValue:"654321" forKey:@"d:ApnsDeviceToken"];
NSError* error = nil;
BOOL isSuccess = [appSettings setConfigProperty:props
error:&error];
if (!error)
{
// Further processing
}
```

Usage

Returns BOOL whether successfully updated or not.

SMPClientConnection class

This class consists of methods used to interact with the underlying client to register connection settings.

Syntax

`public class  SMPClientConnection`

Remarks

The application can initialize/set connection by providing hostname, port, applicationID, domain and security configuration.

*initializeWithAppID:domain:secConfiguration:(NSString *, NSString *, NSString *) method*

Gets the instance/object of the client connection class.

Syntax

`public static virtual SMPClientConnection * initializeWithAppID:domain:secConfiguration: (NSString *` **applicationID**`, NSString *` **domain**`, NSString *` **secConfig**`)`

Parameters

- **applicationID –** Application Id of the application trying to establish the connection
- **domain –** server domain
- **securityConfig –** The name of the security configuration authenticating the user against a certain authentication provider

Returns

Returns the instance of the class SMPClientConnection

Examples

- **Example**

```
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
```

Usage

Returns the instance of the class SMPClientConnection

setClientIdentityCertificate:(SecIdentityRef) method

Sets the identity reference (X.509 certificate + private key) for this request; used for secure connections.

Syntax

`public virtual void setClientIdentityCertificate:(SecIdentityRef` **secIdenref**`)`

Parameters

- **secIdenref –** identity reference (X.509 certificate + private key)

Examples

- **Example**

```
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
[clientConn setClientIdentityCertificate:@"secIdenref"];
[clientConneciton setConnectionProfileWithHost:@"<server_host>"
port:@"<server_port>" farm:nil relayServerUrlTemplate:nil
enableHTTP:YES];
```

*setConnectionProfileWithHost:port:farm:relayServerUrlTemplate:enableHTTP:(NSString *, NSString *, NSString *, NSString *, BOOL) method*

Provides connection properties to the connection class before registration.

Syntax

`public virtual BOOL setConnectionProfileWithHost:port:farm:relayServerUrlTemplate:enableHTTP: (NSString *` **hostName**`, NSString *` **portNumber**`, NSString *` **farmId**`, NSString *` **urlSuffix**`, BOOL` **enableHttp**`)`

Parameters

- **hostName –** Host name of the server
- **portNumber –** Port number of the server with which the client talks
- **enableHttp –** Http protocol to be used

Returns

Returns a BOOL indicating if the operation is successful or not.

Examples

- **Example**

```
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
[clientConneciton setConnectionProfileWithHost:@"10.66.177.94"
port:@"8000" farm:nil relayServerUrlTemplate:nil
enableHTTP:YES];In case of relay server
[clientConneciton
setConnectionProfileWithHost:"relayserver.sybase.com" port:@"80"
farm:@"relay.farm" relayServerUrlTemplate:@"/ias_relay_server/
client/rs_client.dll" enableHTTP:YES];
```

Usage

Provide port as nil if there is no port.

Returns a BOOL indicating if the operation is successful or not.

*setConnectionProfileWithUrl:(NSString *) method*

Provides connection properties to the connection class before registration.

Syntax

`public virtual BOOL setConnectionProfileWithUrl: (NSString *` **hostPort**`)`

Parameters

- **hostPort –** Host name of the server : Port number of the server with which the client talks

Returns

Returns a BOOL indicating if the operation is successful or not.

Examples

- **Example**

```
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
[clientConn setConnectionProfileWithUrl:@"http://
10.66.177.94:8000"];In case of relay server
[clientConn setConnectionProfileWithUrl:"http://
relayserver.sybase.com:80/ias_relay_server/client/rs_client.dll/
relay.farm"];
```

Usage

Returns a BOOL indicating if the operation is successful or not.

setTraceLevel:(E2ELevel) method

Set the trace level, default is E2EMEDIUM.If tracing is enabled, "SAP-PASSPORT" and "X-CorrelationID" headers are set and filled with values to enable SAP Passport functionality.

Syntax

`public virtual void setTraceLevel: (E2ELevel` **value_in**`)`

Examples

- **Example**

```
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
[clientConneciton setConnectionProfileWithHost:@"<server_host>"
port:@"<server_port>" farm:nil relayServerUrlTemplate:nil
enableHTTP:YES];
//On board user
//...
[clientConn setTraceLevel:E2EHIGH];
[clientConn startTrace];
//Request response
[clientConn stopTrace];
NSError* error = nil;
[clientConn uploadTraceWithUserName:"username"
passWord:@"password" error:&error];
if (!error)
{
// Further processing
}
```

startTrace() method

Enables the tracing and starts generating Business Transaction XML.

Syntax

```
public virtual void startTrace ()
```

Examples

- **Example**

```
{
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
[clientConneciton setConnectionProfileWithHost:@"<server_host>"
port:@"<server_port>" farm:nil relayServerUrlTemplate:nil
enableHTTP:YES];
//On board user
//...
[clientConn startTrace];
}
(NSException *e){
// catch exception here
}
```

Usage

Before calling the startTrace API, the corresponding application conection id should be selected using domain->logs->settings->properties->Application Connection->Select connection id.

stopTrace() method

Disables the tracing and stops generating BTX.

Syntax

```
public virtual void stopTrace ()
```

Examples

- **Example**

```
{
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
[clientConneciton setConnectionProfileWithHost:@"<server_host>"
port:@"<server_port>" farm:nil relayServerUrlTemplate:nil
enableHTTP:YES];
//On board user
//...
[clientConn startTrace];
//Request response
[clientConn stopTrace];
}
(NSException *e){
// catch exception here
}
```

*uploadTraceWithUserName:passWord:error:(NSString *, NSString *, NSError **) method*

Uploads the generated BTX to solution manager server via SAP Mobile Platform.

Syntax

`public virtual void uploadTraceWithUserName:passWord:error:(NSString *` **username**`, NSString *` **password**`, NSError **` **error**`)`

Parameters

- **userName –** User Name that will be authorised by a predefined security provider
- **password –** Password for that user
- **error : –** A double pointer to the error object in case the operation results in an error This API will upload the generated BTX to Solution manager .If fails will throw respective error.

Examples

- **Example**

```
SMPClientConnection* clientConn = [SMPClientConnection
initializeWithAppID:"application ID" domain:@"domain"
secConfiguration:@"secconfig"];
[clientConneciton setConnectionProfileWithHost:@"<server_host>"
port:@"<server_port>" farm:nil relayServerUrlTemplate:nil
enableHTTP:YES];
//On board user
//...
[clientConn startTrace];
//Request response
[clientConn stopTrace];
NSError* error = nil;
[clientConn uploadTraceWithUserName:"username"
passWord:@"password" error:&error];
if (!error)
{
// Further processing
}
```

SMPUserManager class
This class consists of methods to register or de-register a user.

Syntax
```
public class  SMPUserManager
```

Remarks
All the function calls in this class have various ways of provisioning settings on to the client and register the user with the help of these settings. You can check if a user is registered and delete the user as required.

Settings Exchange Error Codes

- 70002 - Password Policy Not Enabled.
- 70003 - Error Fetching Push End Point.
- 70004 - Empty Parameters Passed.
- 70005 - Non Editable Properties. This Error is particularly thrown for setConfiguration Property API where properties set are not editable.
- 70006 - Customization Bundle Data Fetch Error.
- 70007 - Customization Bundle Not Available.

User Registration Error Codes

- 70001 - Application Connection Identifier is null. User is not registered.
- 70008 - Parser Error.
- 71000 - SMPInternalError. Error occurred in Rest Client.

*deleteUser:password:error:(NSString *, NSString *, NSError **) method*
Static Method which deletes the application connection/user with which the application has established a connection.

Syntax
```
public virtual BOOL  deleteUser:password:error: (NSString *
userName, NSString * passWord, NSError ** error)
```

Parameters

- **appConID –** Application Connection Id of the user to be deleted
- **userName –** User Name that will be authorised by a predefined security provider
- **password –** Password for that user
- **error –** A double pointer to the error object that the application will pass to the function call and which will be populated by the function call in case of any errors within.

Returns

A BOOL is returned indicating if the user has been deleted or not.

Examples

- **Example**

```
NSError* error = nil;
if([userManager deleteUser:"username" password:@"password"
error:&error];)
{
// Further Processing
}
else
{
// Further Error processing using error object
}
```

Usage

A BOOL is returned indicating if the user has been deleted or not.

getAuthChallengeSelector() method

Getter for Authentication Challenge Selector.

Syntax

```
public static virtual SEL getAuthChallengeSelector ()
```

Returns

Custom Selector set by client as as SEL object.

Usage

This is a class method to fetch the Authentication Challenge Selector.

Custom Selector set by client as as SEL object.

getCaptchaChallengeDelegate() method

Getter for the Captcha Challenge Listener.

Syntax

```
public static virtual id< SMPCaptchaChallengeDelegate >
getCaptchaChallengeDelegate ()
```

Returns

Instance of the listener object.

Usage

This is a class method to fetch the delegate object.

Instance of the listener object.

getDelegate() method

Getter for the user manager listener.This is a class method to fetch the user manager listener object.

Syntax

```
public static virtual id< SMPUserManagerDelegate >
getDelegate ()
```

Returns

Instance of the listener object.

Usage

Instance of the listener object.

*initializeWithConnection:(SMPClientConnection *) method*

Gets the instance/object of the SMPUserManager class.

Syntax

```
public static virtual SMPUserManager *
initializeWithConnection: ( SMPClientConnection * clientConn)
```

Parameters

- **clientConn –** SMPClientConnection object

Returns

Returns the instance of the class SMPUserManager

Examples

- **Example**

  ```
  SMPUserManager* userManager = [SMPUserManager
  initializeWithConnection:clientConn];
  ```

Usage

Returns the instance of the class SMPUserManager

*registerUser:password:captchaText:error:isSyncFlag:(NSString *, NSString *, NSString *, NSError **, BOOL) method*

Static method which registers the client with the server by automatically creating a user with the help of a pre-defined authentication mechanism along with Captcha Text.

Syntax

```
public virtual BOOL
registerUser:password:captchaText:error:isSyncFlag: (NSString
* userName, NSString * loginPassword, NSString * captchaText,
NSError ** error, BOOL isSynchronous)
```

Parameters

- **userName –** User Name that will be authorised by a predefined security provider
- **loginPassword –** Password for that user
- **captchaText –** Captcha Text provided by the user shown in the image
- **error –** A double pointer to the error object that the application will pass to the function call and which will be populated by the function call in case of any errors within.
- **isSynchronous –** A flag indicating whether the user registration has to happen synchronously or in an asynchronous manner

Returns

Returns a BOOL indicating if the user registration has been successful. Significant in case of synchronous registration only.

Examples

- **Example**

```
NSError* error = nil;
[userManager registerUser:"username" password:@"password"
captchaText:@"captchText" error:&error isSyncFlag:NO])
```

Usage

Returns a BOOL indicating if the user registration has been successful. Significant in case of synchronous registration only.

*registerUser:password:error:isSyncFlag:(NSString *, NSString *, NSError **, BOOL) method*

Registers the client with the server by automatically creating a user with the help of a pre-defined authentication mechanism.

Syntax

public virtual BOOL registerUser:password:error:isSyncFlag:(NSString * **userName**, NSString * **loginPassword**, NSError ** **error**, BOOL **isSynchronous**)

Parameters

- **userName –** User Name that will be authorised by a predefined security provider
- **loginPassword –** Password for that user
- **error –** A double pointer to the error object that the application will pass to the function call and which will be populated by the function call in case of any errors within.
- **isSynchronous –** A flag indicating whether the user registration has to happen synchronously or in an asynchronous manner

Returns

Returns a BOOL indicating if the user registration has been successful. Significant in case of synchronous registration only.

Examples

- **Example**

```
Synchronous Registration
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];
NSError* error = nil;
if([userManager registerUser:"username" password:@"password"
error:&error isSyncFlag:YES])
{
// Further processing
} else
{
// Error handling using the error object defined above
}Asynchronous Registration
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];
NSError* error = nil;
[SMPUserManager setDelegate:self]; // The class implementing the
listener methods should adhere to <SMPUserManagerDelegate>
protocol
[userManager registerUser:"username" password:@"password"
error:&error isSyncFlag:NO];
```

Usage

Returns a BOOL indicating if the user registration has been successful. Significant in case of synchronous registration only.

setAuthChallengeSelector:(SEL) method
Setter For the Authentication Challenge.

Syntax

`public static virtual void setAuthChallengeSelector: (SEL` **selector**`)`

Examples

- **Example**

```
// register the client as delegate for this request
[SMPUserManager setAuthChallengeSelector:
(authenticationNeededForRequest:)];
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];
[userManager registerUser:"username" password:@"password"
error:&error isSyncFlag:NO];
...
}
-(void)authenticationNeededForRequest:(SDMHttpRequest*)request
{
//
}
```

Usage

Clients can chose to register their custom selectors to be invoked.

setCaptchaChallengeDelegate:(id< SMPCaptchaChallengeDelegate >) method
Setter for the Captcha Challenge listener.This is a class method to register for the Captcha Challenge Listener.

Syntax

`public static virtual void setCaptchaChallengeDelegate: (id< SMPCaptchaChallengeDelegate >` **delegate**`)`

Parameters

- **listener –** A listener object whose definition adheres to the <SMPCaptchaChallengeDelegate> protocol that has to be implemented by the application to receive Captcha Challenge

Examples

- **Example**

  ```
  // register the client as delegate for this request
  [SMPUserManager setCaptchaChallengeDelegate:self];
  SMPUserManager* userManager = [SMPUserManager
  initializeWithConnection:clientConn];
  [userManager registerUser:"username" password:@"password"
  error:&error isSyncFlag:NO];
  ...
  }
  -(NSString*)didReceiveCaptchaChallenge:
  (NSString*)base64ImageString
  {
  // Return text String input from user.
  }
  ```

setDelegate:(id< SMPUserManagerDelegate >) method

Setter for the user manager listener.This is a class method to register the user manager listener .

Syntax

`public static virtual void setDelegate: (id< SMPUserManagerDelegate > `**delegate**`)`

Parameters

- **listener –** A listener object whose definition adheres to the <SMPUserManagerDelegate> protocol that has to be implemented by the client to get the pushed data

Examples

- **Example**

  ```
  // register the client as delegate for this request
  [SMPUserManager setDelegate:self];
  SMPUserManager* userManager = [SMPUserManager
  initializeWithConnection:clientConn];
  [userManager registerUser:"username" password:@"password"
  error:&error isSyncFlag:NO];
  ...
  }
  ```

SMPCaptchaChallengeDelegate protocol

This protocol has to be adhered by any class that implements the captcha challenge delegates.

Syntax

`public protocol SMPCaptchaChallengeDelegate`

*didReceiveCaptchaChallenge:(NSString *) method*

The captcha challenge delegate which will be called by the underlying framework when captcha challenge is required during the user registration.

Syntax

`public virtual NSString * didReceiveCaptchaChallenge: (NSString *` **base64ImageString**`)`

Examples

- **Example**

```
-(IBAction)buttonsPressed:(UIButton*)button{[SMPUserManager
setCaptchaChallengeDelegate:self];
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];}-
(NSString*)didReceiveCaptchaChallenge:base64ImageString
{
// Return text String input from user.
}
```

Usage

This is supported in case of SAP Mobile Platform, enterprise edition, cloud version onboarding only.

SMPUserManagerDelegate protocol

This protocol has to be adhered by any class that implements the success and failure notification delegates and this class has to be passed to the delegate variable of the user manager class object.

Syntax

`public protocol  SMPUserManagerDelegate`

*userRegistrationFailed:(SMPUserManager *) method*

The failure listener delegate which will be called by the underlying framework when the user registration has failed and the reason for failure will be propagated through an error object.

Syntax

`public virtual void userRegistrationFailed:( SMPUserManager *` **userManager**`)`

Examples

- **Example**

```
[SMPUserManager setDelegate:self];
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];


-(void)userRegistrationFailed:userManager
{
// Error handling using the error object
}
```

Usage

This is true in case of asynchronous user registration only.

*userRegistrationSuccessful:(SMPUserManager *) method*

The success listener delegate which will be called by the underlying framework when the user registration is successful.

Syntax

`public virtual void userRegistrationSuccessful:`
`( SMPUserManager *` **userManager**`)`

Examples

- **Example**

```
-(IBAction)buttonsPressed:(UIButton*)button{[SMPUserManager
setDelegate:self];
SMPUserManager* userManager = [SMPUserManager
initializeWithConnection:clientConn];}-
(void)userRegistrationSuccessful:userManager
{
// Successful post-processing
}
```

Usage

This is true in case of asynchronous user registration only.

Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)

Use `setRequestType` method in `SDMRequestBuilder` class to set the request-response using Messaging Channel or REST SDK.

To enable request-response using REST SDK, set request type to `HTTPRequestType` in the `SDMRequestBuilder` class:

```
[SDMRequestBuilder setRequestType:HTTPRequestType];
```

To enable request-response using Messaging Channel, do not set any value for request type in `SDMRequestBuilder` class. The default is set to use Messaging Channel.

See also

- *Development Task Flow Using REST SDK (HTTP Channel)* on page 41

Deploying Applications to Devices

Complete steps required to deploy mobile applications to devices.

1. *Signing*

   Code signing is required for applications to run on physical devices.
2. *Configuring Apple Push Notification Service*

   Use Apple Push Notification Service (APNS) to push notifications from SAP Mobile Server to the iOS application. Notifications might include badges, sounds, or custom text alerts. Device users can use Settings to customize which notifications to receive or ignore.
3. *Provisioning an Application for Apple Push Notification Service*

   Use Apple Push Notification Service (APNS) to push notifications from SAP Mobile Server to the iOS application. Notifications might include badges, sounds, or custom text alerts. Device users can use Settings to customize which notifications to receive or ignore.
4. *Preparing Applications for Deployment to the Enterprise*

   After you have created your client application, you must sign your application with a certificate from Apple, and deploy it to your enterprise.

See also

- *Testing Applications* on page 83

Signing

Code signing is required for applications to run on physical devices.

Configuring Apple Push Notification Service

Use Apple Push Notification Service (APNS) to push notifications from SAP Mobile Server to the iOS application. Notifications might include badges, sounds, or custom text alerts. Device users can use Settings to customize which notifications to receive or ignore.

Prerequisites

Perform these prerequisites in the Apple Developer Connection Portal:

- Register for the iPhone Developer Program as an enterprise developer to access the Developer Connection portal and get the certificate required to sign applications.
- Create an App ID and ensure that it is configured to use Apple Push Notification Service (APNS).
- Create and download an enterprise APNS certificate that uses Keychain Access in the Mac OS. The information in the certificate request must use a different common name than the development certificate development teams might already have. This is because the enterprise certificate also creates a private key, which must be distinct from the development key. This certificate must also be imported as a login keychain and not a system keychain and the developer should validate that the certificate is associated with the key in the Keychain Access application. Get a copy of this certificate.

  Note: A new 2048-bit Entrust certificate needed for APNS.

  Apple uses a 2048-bit root certificate from Entrust, which provides a more secure connection between SAP Mobile Server and APNS. This certificate comes with the Windows OS, and is upgraded automatically with Windows Update, if it is enabled. This information is not part of the procedure that documents APNS support.

  If Windows Update is disabled, you must manually download and install the certificate (entrust_2048_ca.cer). Go to *https://www.entrust.net/downloads/root_index.cfm*. For help on installing the certificate, see *http://www.entrust.net/knowledge-base/technote.cfm?tn=8282*.

- Create an enterprise provisioning profile and include the required device IDs with the enterprise certificate. The provisioning profile authorizes devices to use applications you have signed.
- Create the Xcode project ensuring the bundle identifier corresponds to the bundle identifier in the specified App ID. Ensure you are informed of the "Product Name" used in this project.
- Use the APNS initialization code.

Developers can review complete details in the *iPhone OS Enterprise Deployment Guide* available on the Apple Developers Website.

Task

Each application that supports Apple Push Notifications must be listed in SAP Control Center with its certificate and application name. You must perform this task for each application.

1. Confirm that the IT department has opened ports 2195 and 2196, by executing:

   ```
   telnet gateway.push.apple.com 2195
   telnet feedback.push.apple.com 2196
   ```

   If the ports are open, you can connect to the Apple push gateway and receive feedback from it.
2. Upload the APNS certificate to SAP Control Center:

   a) In the navigation pane, click **Applications**.
   b) In the administration pane, click the **Applications** tab.
   c) Select the application for which you want to enable APNS, and click **Properties**.
   d) Click the **Push Configurations** tab and click on **Add**.
   e) Configure all required properties, including the corresponding password and upload the certificate. See *APNS Native Notification Properties* in *SAP Control Center for SAP Mobile Platform* online help.

3. Deploy the iOS application with an enterprise distribution provisioning profile to users' iOS devices.
4. Verify that the APNS-enabled iOS device is set up correctly:
   a) In SAP Control Center, ensure the user has already activated the application and is connected to the SAP Mobile Server, by looking for the corresponding entry in **ApplicationsApplication Connections**.
   b) Validate that in the Application Connection ID, the application name appears correctly at the end of the string.
   c) Select the user and click **Properties**.
   d) Check that the *APNS Device Token* contains a value. This indicates that a token has passed successfully following a successful application activation
5. Verify that native notification is enabled for the user:
   a) Select the user name and click **Properties**.
      - For Application Settings, ensure the **Notification Mode** property is set to either **Only native notifications** or **Online/ payload push with native notification**.
      - For Apple Push Notifications, ensure the **Enabled** property is set to **True**.
6. Test the environment by initiating an action that results in a new message being sent to the client.

   If you have verified that both device and server can establish a connection to the APNS gateway, the device receives notifications and messages from the SAP Mobile Server. If you configured **Online/ payload push with native notification**, allow a few minutes for the delivery. If the device is offline and the message is pending in messaging queue, SAP Mobile Server triggers the native push notification mechanism to send the Pending Items to the device via APNS. See *Reviewing the Pending Items Count for Messaging Applications* in *System Administration*.

   Note: Notifications require a connection to APNS on port 5223. This port is not always routed through the firewall on corporate wireless networks.
7. To troubleshoot APNS, use the `SMP_HOME\Servers\SAP Mobile Server\logs\server` log file.

Provisioning an Application for Apple Push Notification Service

Use Apple Push Notification Service (APNS) to push notifications from SAP Mobile Server to the iOS application. Notifications might include badges, sounds, or custom text alerts. Device users can use Settings to customize which notifications to receive or ignore.

Prerequisites

Ensure that you have a P12 certificate generated using the developer certificate (for code signing and deployment on to the device), and the private key of the certificate on your local machine.

Task

Each application that supports Apple Push Notifications must be listed in SAP Control Center with its certificate and application name. You must perform this task for each application.

1. Confirm that the IT department has opened ports 2195 and 2196, by executing:

   ```
   telnet gateway.push.apple.com 2195
   telnet feedback.push.apple.com 2196
   ```

   If the ports are open, you can connect to the Apple push gateway and receive feedback from it.
2. Upload the APNS certificate to SAP Control Center:
   a) In the navigation pane, click **Applications**.
   b) In the administration pane, click the **Applications** tab.
   c) Select the application for which you want to enable APNS, and click **Properties**.
   d) Click the **Push Configurations** tab and click on **Add**.
   e) Configure all required properties, including the corresponding password and upload the certificate. See *APNS Native Notification Properties* in *SAP Control Center for SAP Mobile Platform* online help.
3. Deploy the iOS application with an enterprise distribution provisioning profile to users' iOS devices.
4. Instruct users to use iTunes to install the application and profile, and how to enable notifications. In particular, device users must:
   - Download the application from the App Store.
   - In the iPhone Settings app, slide the **Notifications** control to **On**.
5. Verify that the APNS-enabled iOS device is set up correctly:
   a) In SAP Control Center, ensure the user has already activated the application and is connected to the SAP Mobile Server, by looking for the corresponding entry in **ApplicationsApplication Connections**.
   b) Validate that in the Application Connection ID, the application name appears correctly at the end of the string.

   c) Select the user and click **Properties**.
   d) Check that the *APNS Device Token* contains a value. This indicates that a token has passed successfully following a successful application activation

6. Test the environment by initiating an action that results in a new message being sent to the client.

   If you have verified that both device and server can establish a connection to APNS gateway, the device will receive notifications and messages from the SAP Mobile Server, including workflow messages, and any other messages that are meant to be delivered to that device. Allow a few minutes for the delivery or notification mechanism to take effect and monitor the data in the **Pending Items** column in the **Application Users** area to see that the value increases appropriately for the applications.
7. To troubleshoot APNS, use the `SMP_HOME\Servers\SAP Mobile Server \logs\server` log file.

Preparing Applications for Deployment to the Enterprise

After you have created your client application, you must sign your application with a certificate from Apple, and deploy it to your enterprise.

Note: Review complete details in the *iPhone OS Enterprise Deployment Guide* available on the Apple Developers Website, and *About Your First App Store Submission* at *https://developer.apple.com/library/ios/#documentation/ToolsLanguages/Conceptual/YourFirstAppStoreSubmission/AboutYourFirstAppStoreSubmission/AboutYourFirstAppStoreSubmission.html#//apple_ref/doc/uid/TP40011375-CH1-SW1*.

1. Sign up for the iOS Developer Program, which gives you access to the Developer Connection portal. Registering as an enterprise developer gets you the certificate you need to sign applications.
2. Create a certificate request on your Mac through Keychain.
3. Log in to the Developer Connection portal.
4. Upload your certificate request.
5. Download the certificate to your Mac. Use this certificate to sign your application.
6. Create an AppID.

   Verify that your `info.plist` file has the correct AppID and application name. Also, in Xcode, right-click **Targets** > **<your_app_target>** and select **Get Info** to verify the AppID and App name.
7. Create an enterprise provisioning profile and include the required device IDs with the enterprise certificate. The provisioning profile authorizes devices to use applications you have signed.
8. Create an Xcode project ensuring the bundle identifier corresponds to the bundle identifier in the specified App ID. Ensure you are informed of the "Product Name" used in this project.

Testing Applications

Test native applications on a device or simulator.

For additional information about testing applications, see these topics in the Mobile Application Life Cycle collection:

- *Recommended Test Methodologies*
- *Best Practices for Testing Applications on a Physical Device*

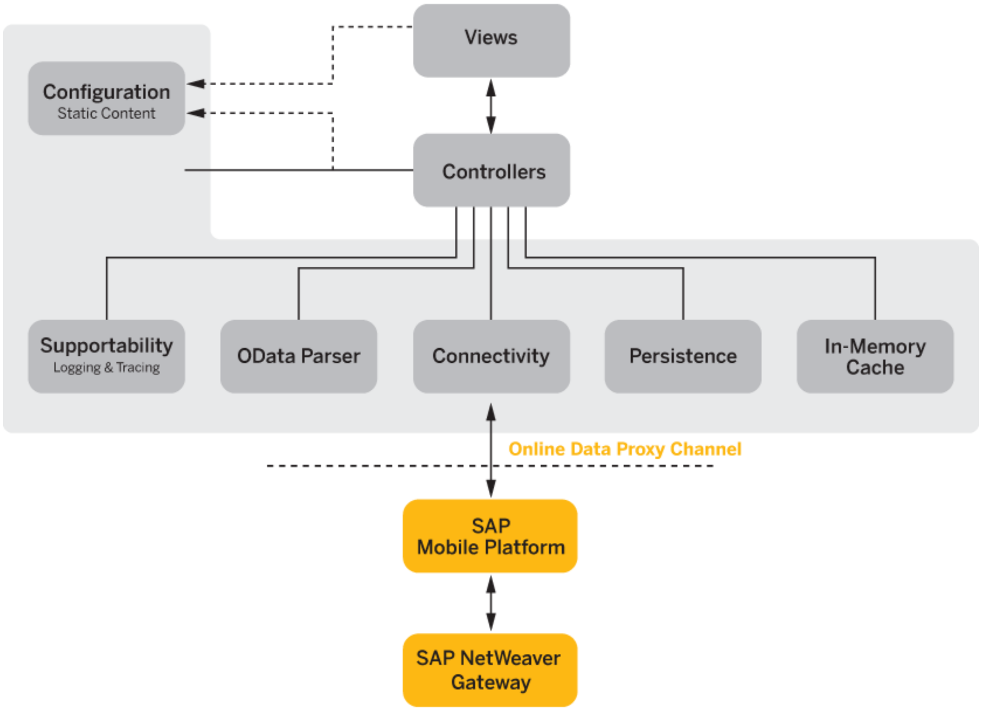
See also

- *Deploying Applications to Devices* on page 78

OData SDK Components and APIs

The iOS OData SDK provides the means to easily build an app which relies on the OData protocol and the additions made by SAP.

OData SDK - iOS
The following figure shows the main components of the OData SDK on iOS.



The iOS version of the OData SDK is presented as static libraries and header files. (Custom dynamic libraries are not allowed on iOS.)

The OData SDK for iOS includes a set of core iOS libraries acting independently from each other. Each core library has well-defined responsibilities and provides APIs for OData parsing, caching, persistence, keychain, certificate management, and so on.

The full list of APIs and their descriptions are available after the installation of SAP Mobile Platform at the following location within your installation folder:*`SMP_HOME`* `MobileSDK<Version>\OData\iOS\docs`.

The libraries are provided in binary form as `.a` files, along with the public headers containing the APIs and the input/output structures. As a prerequisite, the public headers and the libraries must be available as separate binaries for release and debug, or merged using the **lipo** tool.

SDMParser

The SDMParser library provides APIs to convert OData XML payloads to native Objective-C objects and structures (arrays, dictionaries).

List of Features

- OData XML or OData with SAP extensions XML (including inlined content) parsing and conversion to Objective-C objects
- URL template retrieval from open search description XMLs
- OData XML composition (create update scenario), also with SAP extensions
- OData error XML parsing
- Function import support
- Generates subscription XMLs
- Media Link Entries
- Convenient C-style APIs
- Action Link Support
- ETag Support

SDMParser Public APIs

Provides public interfaces of SDM parser.

SDMParser Public APIs

```
SDMODataServiceDocument* sdmParseODataServiceDocumentXML(NSData*
const content_in)
SDMODataSchema* sdmParseODataSchemaXML(NSData* const content_in,
SDMODataServiceDocument* const serviceDocument)
NSMutableArray* sdmParseODataEntriesXML(NSData* const content_in,
const SDMODataEntitySchema* const entitySchema, const
SDMODataServiceDocument* const serviceDocument)
SDMODataError* sdmParseODataErrorXML(NSData* const content_in)
NSMutableArray* sdmParseFunctionImportResult(NSData* const
content_in, const SDMODataFunctionImport* const functionImport)
```

```
SDMOpenSearchDescription* sdmParseOpenSearchDescriptionXML(NSData*
const content_in)
SDMODataEntryXML* sdmBuildODataEntryXML (const SDMODataEntry *const
entry, const enum TEN_ENTRY_OPERATIONS operation, const
SDMODataServiceDocument *const serviceDocument, const BOOL
serializeInlinedEntries)
SDMODataFeedXML* sdmBuildODataFeedXML (NSArray *const entries, const
enum TEN_ENTRY_OPERATIONS operation, const SDMODataServiceDocument
*const serviceDocument, const BOOL serializeInlinedEntries)
(NSString *)getEtag
```

Technical Details

The listed C-style parser APIs are provided for convenience. You can choose to instantiate the dedicated parser classes. As a reference, the following code excerpt shows how the C-style APIs wrap the parser calls:

```
/**
 * Parses the service document XML and converts it to an Obj-C
service document object.
 */
SDMODataServiceDocument* sdmParseODataServiceDocumentXML(NSData*
const content_in) {
    SDMODataServiceDocumentParser* svcDocParser =
[[[SDMODataServiceDocumentParser alloc] init] autorelease];
    [svcDocParser parse: content_in];

    return svcDocParser.serviceDocument;
}


/**
 *    Parses and matches the schema with the service document and its
collections. The function returns the same
 *  schema pointer as it can already be found in the serviceDocument.
 */
SDMODataSchema* sdmParseODataSchemaXML(NSData* const content_in,
SDMODataServiceDocument* const serviceDocument) {
    if (!serviceDocument)
        //@throw [[[SDMParserException alloc] initWithName:
@"NoServiceDocument" reason: @"No service document was provided"
userInfo: nil] autorelease];
        @throw [[[SDMParserException alloc] initWithError:
ParserNoServiceDocument detailedError: @"No service document was
provided"] autorelease];

    SDMODataMetaDocumentParser* metaDocParser =
[[[SDMODataMetaDocumentParser alloc] initWithServiceDocument:
serviceDocument] autorelease];
    [metaDocParser parse: content_in];

    return serviceDocument.schema;
}


/**
```

```
 * Parses a feed or entry XML and returns an array of parsed entry/
entries.
 * Any "inlined"entries or feed(s) will be parsed when service
document is passed to the function. If "inlined" feed(s) or entries
 * should not be returned pass nil in the service document parameter.
 */
NSMutableArray* sdmParseODataEntriesXML(NSData* const content_in,
const SDMODataEntitySchema* const entitySchema, const
SDMODataServiceDocument* const serviceDocument) {
    if (!entitySchema)
        //@throw [[[SDMParserException alloc] initWithName:
@"NoEntitySchema" reason: @"No entity schema was provided" userInfo:
nil] autorelease];
        @throw [[[SDMParserException alloc] initWithError:
ParserNoEntitySchema detailedError: @"No entity schema was
provided"] autorelease];

    SDMODataDataParser* dataParser = [[[SDMODataDataParser alloc]
initWithEntitySchema: entitySchema andServiceDocument:
serviceDocument] autorelease];
    [dataParser parse: content_in];

    return dataParser.entries;
}


/**
 * Parses an OData error payload XML
 * @see SDMODataError
 */
SDMODataError* sdmParseODataErrorXML(NSData* const content_in) {
    SDMODataErrorXMLParser* errorParser = [[[SDMODataErrorXMLParser
alloc] init] autorelease];
    [errorParser parse: content_in];

    return errorParser.odataError;
}

/**
 * Parses the result payload XML of a function import.
 * @returns Returns an array of entries.
 * @remark Even if the result is not a feed or entry XML, the parser
creates an entity schema out of the return type definition, so
 * application developers can access the returned data in a uniform
way. The supported return types are:
 * - none
 * - EDMSimpleType        (for example: ReturnType="Edm.Int32"), the
generated "entity" schema will be "element" with type Edm.Int32
 * - ComplexType        (for example:
ReturnType="NetflixCatalog.Model.BoxArt")
 * - Collection of an EDMSimpleType (for example:
ReturnType="Collection(Edm.String)")
 * - Collection of a ComplexType    (for example:
ReturnType="Collection(NetflixCatalog.Model.BoxArt)")
 * - Entry    (for example ReturnType="NetflixCatalog.Model.Title"
EntitySet="Titles")
```

```
 * - Feed   (for example
ReturnType="Collection(NetflixCatalog.Model.Title)"
EntitySet="Titles")
 */
NSMutableArray* sdmParseFunctionImportResult(NSData* const
content_in, const SDMODataFunctionImport* const functionImport) {
    SDMFunctionImportResultParser* fiParser =
[[[SDMFunctionImportResultParser alloc] initWithFunctionImport:
functionImport] autorelease];
    [fiParser parse: content_in];

    return fiParser.entries;
}


/**
 * Parses an XML that contains Open Search Description
 * The parsed data is returned in an SDMOpenSearchDescription typed
object.
 */
SDMOpenSearchDescription* sdmParseOpenSearchDescriptionXML(NSData*
const content_in) {
    SDMOpenSearchDescriptionXMLParser* osdParser =
[[[SDMOpenSearchDescriptionXMLParser alloc] init] autorelease];
    [osdParser parse: content_in];

    return osdParser.openSearchDescription;
}
```

The SDMParser library communicates error conditions to the client via the dedicated `SDMParserException` exception class. Whenever a mandatory attribute is missing, the parser throws an exception.

The caller is responsible for error handling; this includes fetching the details included in the exception, logging information meant for debugging purposes, displaying a localized alert message, and providing a resolution or stopping the application flow.

SDMParser Components

Provides a list of various components in SDM parser.

The Service Document Component

Root object. Contains the schema object, the function imports, the document language, base URL (if any) and the server type.

```
SDMOdataServiceDocument

-(enum TEN_SERVER_TYPES)getServerType
-(NSString*)getDocumentLanguage
-(NSString*)getBaseUrl
-(SDMODataSchema*)getSchema
-(NSMutableDictionary*)getFunctionImports
```

The Schema Component

The schema contains workspaces and helper methods to work with collections via workspaces.

```
SDMODataSchema

-(NSArray*) getWorkspacesBySemantic:(const enum
TEN_WORKSPACE_SEMANTICS)workspaceSemantic
-(SDMODataCollection*) getCollectionByName:(NSString*
const)collectionName
-(SDMODataCollection*) getCollectionByName:(NSString*
const)collectionName workspaceOfCollection:
(SDMODataWorkspace**)workspaceOfCollection
```

The Workspace Component

A workspace can contain 0 up to n collections. Each workspace can have a title and a semantic value.

```
SDMODataWorkspace

-(enum TEN_WORKSPACE_SEMANTICS) getSemantic
-(NSString*) getTitle
-(NSMutableDictionary*) getCollections
```

The Collection Component

Represents one parsed collection.

```
SDMODataCollection

-(id) initWithName:(NSString* const)newName
-(BOOL) isCreatable
-(BOOL) isUpdatable
-(BOOL) isDeletable
-(BOOL) isTopLevel
-(BOOL) doesRequireFilter
-(BOOL) hasMedia
-(SDMODataLink*) getSubscriptionLink
-(int) getContentVersion
-(enum TEN_COLLECTION_SEMANTICS) getSemantic
-(uint8_t) getFlags
-(NSString*) getName
-(NSString*) getTitle
-(NSString*) getMemberTitle
-(NSMutableArray*) getIcons
-(NSMutableArray*) getLinks
-(int) getDisplayOrder
-(SDMODataEntitySchema*) getEntitySchema
-(SDMOpenSearchDescription*) getOpenSearchDescription
```

The Entity Schema Component

An instance of the `EntitySchema` class stores the root of the structure of the given collection with constraints. The entity schema class also provides helper functions to order the visible fields of a collection and the navigation map that maps navigation names to collection names.

```
SDMODataEntitySchema

-(id) init
-(int) getContentVersion
-(uint16_t) getFlags
-(SDMODataPropertyInfo*) getRoot
-(NSMutableDictionary*) getNavigationMap
-(NSArray* const) getVisibleInListPathsInOrder
-(NSArray* const) getVisibleInDetailPathsInOrder
```

The Property Info Component

A property info instance stores the name, type and all constraints of a property, but does not store property values.

```
SDMODataPropertyInfo

-(id) initWithName:(NSString* const)propName andPropEdmType:(const
enum TEN_EDM_TYPES)propEdmType
-(BOOL) isNullable
-(BOOL) isKey
-(BOOL) isCreatable
-(BOOL) isUpdatable
-(BOOL) isFilterable
-(BOOL) isVisibleInList
-(BOOL) isVisibleInDetail
-(BOOL) isSearchable
-(BOOL) isServerGenerated
-(void) addChildPropertyInfo:(const SDMODataPropertyInfo*
const)child
-(SDMODataPropertyInfo* const) getPropertyInfoByPath:(NSString*
const)path
-(NSString*) getName
-(enum TEN_EDM_TYPES) getType
-(uint16_t) getFlags
-(int) getMaxLength
-(enum TEN_PROPERTY_SEMANTICS) getSemantic
-(uint32_t) getSemanticTypes
-(NSString*) getLabel
-(NSString*) getDescription
-(int32_t) getListDisplayOrder
-(int32_t) getDetailDisplayOrder
-(uint8_t) getScale
-(uint8_t) getPrecision
-(NSMutableDictionary*) getChildren
```

The Function Import Component

Function imports can be used to execute back-end functionalities that are not related to collections, or functionalities other than the possible create, update, delete and read operations for collections. An instance of `SMDODataFunctionImport` stores all the information and has all the methods necessary to execute such a back-end functionality.

```
SDMODataFunctionImport

-(id) initWithName:(NSString* const)newName
-(NSString*) getName
-(NSString*) getHttpMethod
-(NSMutableDictionary*) getParameters
-(SDMODataEntitySchema*) getReturnTypeSchema
-(uint8_t) getFlags
-(NSString*) getActionFor
-(NSMutableDictionary*) getWritableParameters
-(NSString*) generateFunctionImportUrl:(NSString* const)baseUrl
parameters:(NSDictionary* const)parameters
```

The Link Component

The OData SDK provides four types of link classes depending on the use case:

- `SDMODataLink`
- `SDMODataRelatedLink` (this class inherits all the methods mentioned at `SDMODataLink`)
- `SDMODataMediaResourceLink` (this class inherits all the methods mentioned at `SDMODataLink`)
- `SDMODataActionLink` (contains the optional parameters of the action and the helper method to assemble the final URL that is required to execute the action)

```
SDMODataLink

-(NSString*) getHRef
-(NSString*) getRel
-(NSString*) getType
-(NSString*) getTitle
-(enum TEN_LINK_SEMANTICS) getSemantic


DMODataRelatedLink

-(NSString*) getTargetCollection


SDMODataMediaResourceLink

-(NSString*) getConcurrencyToken


SDMODataActionLink

-(NSString*) getHttpMethod;
-(NSMutableDictionary*) getDefaultParameterValues;
-(NSDictionary*) getParameters;
-(NSString*) createActionLinkURL:(NSDictionary*)parameters;
```

The Open Search Description Component

An `SDMOpenSearchDescription` instance stores the parsed short name, description and the URL templates for searching data.

```
SDMOpenSearchDescription

-(NSString*) getShortName
-(NSString*) getDescription
-(NSMutableArray*) getUrlTemplates

SDMOpenSearchDescriptionURLTemplate

-(NSString*) getUrlTemplate
-(NSString*) getUrlType
-(NSString*) createUrlWithParameters:(NSDictionary*)parameters
```

The Property Value Objects Component

An instance of the property value object stores a value and its metadata (property info instance). `SDMODataPropertyValueObject` is the base property value class that provides basic validation and value accessors. Derived classes of this class redefine certain methods (for example, validation checks) of the base class and provide methods allowing the library user to access data as typed data instead of string data.

```
SDMODataPropertyValueObject (base class)

-(NSString* const) getHTMLEncodedValue
-(NSString* const) getDefaultValue
-(BOOL) isValid
-(NSString*) getValue
-(void) setValue:(NSString*) value
-(enum TEN_EDM_TYPES) getEdmType
-(const SDMODataPropertyInfo* const) getPropertyInfo
-(BOOL) isValidationDisabled
-(void) setValidationDisabled:(BOOL)validationDisabled
```

ETag Support

An entity tag is one of the several mechanisms that HTTP provides for cache validation and which allows a client to make conditional requests. An ETag is an identifier assigned by a Web Server to a specific version of a resource found at a URL. If the resource content at the URL changes, a new and different ETag is assigned.

Examples

Provides example snippets to understand the use of SDMParser APIs.

ETag Support

This code snippet is used to extract an ETag attribute of an atom-entry using the SDMParser library.

```
NSMutableArray *entries = nil;
//Get all entries in a given collection
```

```
entries = sdmParseODataEntriesXML([request responseData],
collection.entitySchema, nil);
for(int i=0; i<[entries count];i++) {
SDMODataEntry *entry = [entries objectAtIndex:i];
//For each entry get the ETag attribute
NSString *etag = [entry getEtag];
}
```

SDMCache

The SDMCache is a programming interface that provides in-memory cache for quick data access. Its APIs allow adding, removing and searching items stored in the cache. The cache also acts as a central, shared storage, avoiding the need to pass frequently used data between view controllers.

List of Features

- In-memory management of SDMOData-related objects
- Quick data filtering
- Prefix matching and regular expression support (default search method is prefix matching)

SDMCache Public APIs

```
- (void) setCapacity:(unsigned short) value
- (unsigned short) capacity
- (void) clear
- (void) setODataServiceDocument:(SDMODataServiceDocument*)
serviceDocument_in
- (id) initWithServiceDocument:(SDMODataServicedocument*)
serviceDoc_in
+ (id) cacheWithServiceDoc:(SDMODataServicedocument*) serviceDoc_in
- (NSArray*)filterEntriesOfCollection:(NSString*) collectionName_in
forSearchText:(NSString*)searchText_in
- (SDMODataServiceDocument*) getODataServiceDocument
- (BOOL) removeODataServiceDocument
- (void) setODataEntry:(SDMODataEntry*) entry_in byCollection:
(NSString*) collectionName_in
- (SDMODataEntry*) getODataEntryByCollection:(NSString*)
collectionName_in andEntryId:(NSString*) entryId_in
- (void) setODataEntries:(NSArray *) entries_in byCollection:
(NSString*) collectionName_in
- (NSArray*) getODataEntriesByCollection:(NSString*)
collectionName_in
- (NSArray*) getCollectionsByWorkspace:
(SDMDataWorkspace*)workspace_in
- (SDMODataCollection*) getCollectionByName:
(NSString*)collectionName_in
- (BOOL) removeODataEntry:(SDMODataEntry*)entry_in forCollection:
(NSString*) collectionName_in
- (BOOL) removeODataEntriesForCollection:(NSString*)
collectionName_in
- (NSArray*) getAllWorkspaces
- (NSArray*) getWorkspacesBySemantic:(enum TEN_WORKSPACE_SEMANTICS)
- (void) setShouldAutoSaveOnMemoryWarning:(BOOL)flag_in
- (BOOL) shouldAutoSaveOnMemoryWarning
- (void) setRegexSearchEnabled:(BOOL)flag_in
- (BOOL) isRegexSearchEnabled
```

Technical Details

The common methods are defined by the `SDMCaching` protocol. (See also: `SDMCache` default implementation.)

SDMCache Public APIs

SDMCache Public APIs

```
- (void) setCapacity:(unsigned short) value
- (unsigned short) capacity
- (void) clear
- (void) setODataServiceDocument:(SDMODataServiceDocument*)
serviceDocument_in
- (id) initWithServiceDocument:(SDMODataServicedocument*)
serviceDoc_in
+ (id) cacheWithServiceDoc:(SDMODataServicedocument*) serviceDoc_in
- (NSArray*)filterEntriesOfCollection:(NSString*) collectionName_in
forSearchText:(NSString*)searchText_in
- (SDMODataServiceDocument*) getODataServiceDocument
- (BOOL) removeODataServiceDocument
- (void) setODataEntry:(SDMODataEntry*) entry_in byCollection:
(NSString*) collectionName_in
- (SDMODataEntry*) getODataEntryByCollection:(NSString*)
collectionName_in andEntryId:(NSString*) entryId_in
- (void) setODataEntries:(NSArray *) entries_in byCollection:
(NSString*) collectionName_in
- (NSArray*) getODataEntriesByCollection:(NSString*)
collectionName_in
- (NSArray*) getCollectionsByWorkspace:
(SDMDataWorkspace*)workspace_in
- (SDMODataCollection*) getCollectionByName:
(NSString*)collectionName_in
- (BOOL) removeODataEntry:(SDMODataEntry*)entry_in forCollection:
(NSString*) collectionName_in
- (BOOL) removeODataEntriesForCollection:(NSString*)
collectionName_in
- (NSArray*) getAllWorkspaces
- (NSArray*) getWorkspacesBySemantic:(enum TEN_WORKSPACE_SEMANTICS)
- (void) setShouldAutoSaveOnMemoryWarning:(BOOL)flag_in
- (BOOL) shouldAutoSaveOnMemoryWarning
- (void) setRegexSearchEnabled:(BOOL)flag_in
- (BOOL) isRegexSearchEnabled
```

Technical Details

The common methods are defined by the `SDMCaching` protocol. (See also: `SDMCache` default implementation.)

SDMPersistence

The SDMPersistence library provides APIs to persist data to the device's physical storage.

List of Features

- Protected data storage to the device’s filesystem using iOS 4.0 features
- Configurable storage policy
- Stores and loads NSData, SDMCache and objects adopting the NSCoding protocol

SDMPersistence Public APIs

SDMPersistence Public APIs

```
+ (id) instance
- (void) clear
- (NSString*) storeCache:(id<SDMCaching>)cache_in
- (id<SDMCaching>) loadCache:(NSString*) uid_in
- (NSString*) storeData:(NSData*) data_in withId:(NSString*)uid_in
- (NSData*) loadData:(NSString*) uid_in
- (NSString*) storeSerializable:(id<NSCoding>) serializable_in
withId:(NSString*)uid_in
- (id<NSCoding>) loadSerializable:(NSString*) uid_in
- (StoragePolicy) storagePolicy
- setStoragePolicy:(StoragePolicy)policy_in
```

Technical Details

The common methods are defined by the `SDMPersisting` protocol. For builds targeting iOS 4.0+ the default is `FullProtectionStoragePolicy`. (See also: `SDMPersistence` default implementation.)

Consider using the SDMPersistence default implementation rather than implementing a custom persistence functionality.

Encryption and Secure Storage

Encryption, Secure Storage

Starting with iOS 4.0 (iOS 4.2 for iPad devices), data can be persisted in secure form. For builds targeting iOS 4.0+, the default storage policy is fully protected. In older iOS versions, data can only be persisted in unprotected form.

All data is stored in the app’s dedicated filesystem, the so-called sandbox. The app’s sandbox can be accessed exclusively by the app it belongs to. As the sandbox is bound to the app, deleting the app also removes its persisted data. Accessing data on iOS devices is fast and reliable, even when encryption is used.

For encryption, we rely on the Security framework and the dedicated cryptographical hardware available in the supported versions of iPad, iPad 2 and iPhone. Due to the lack of the cryptographical hardware, former iPhone models are not supported. The RSA keys required for the asymmetric key algorithm are retrieved from the app's keychain; if it is not available, they are generated and stored in the keychain during the first API call requiring the keys. For RSA key generation and keychain management, the iOS Security framework APIs are used.

A generic approach for secure data storage has been made available with iOS 4.0. Encryption and decryption of the device's filesystem is managed automatically by the operating system. This behavior is disabled by default, but can be enforced via corporate policy. It is possible to leave out the secure APIs from the library and solely rely on this approach.

Related reading: *http://support.apple.com/kb/HT4175*

WWDC video about secure data storage: *http://developer.apple.com/videos/wwdc/2010/*

SDMConnectivity

The Connectivity library exposes APIs required to set up and start HTTP requests, and retrieve the payloads. For crossing a company firewall for enterprise use cases, you need to use SAP Mobile Platform. Therefore, the connectivity component in the OData SDK offers a connection to SAP Mobile Platform by default. For development and demo purposes, the SDK also provides a possibility to use HTTP or HTTPS.

List of Features

- Synchronous and asynchronous HTTP request handling
- Concurrent request execution
- Continuous downloading and uploading when the app is sent to the background (iOS 4.0+ only)
- Timeout handling
- Supports compressed payload handling
- Notification about various events (failure, completion, authentication requests)
- Runtime switch between `SDMHttpRequest` and `SUPRequest` (SAP Mobile Platform libraries are needed to be linked to the project) through `SDMRequestBuilder`
- ETag support

SDMConnectivity Public APIs

SDMConnectivity Public APIs

Note: The SAP Mobile Platform APIs and their descriptions are available after the installation of SAP Mobile Platform at the following location within your installation folder: `...\UnwiredPlatform\ClientAPI\apidoc`.

```
-(id)initWithURL:(NSURL *)newURL
+(id)requestWithURL:(NSURL *)newURL
-(void) setUserName:(NSString*)username
-(void) setPassword:(NSString*)username
-(void) setClientDelegate:(id)clientDelegate
-(void) setDidFinishSelector:(SEL)didFinishSelector
-(void) setDidFailSelector:(SEL)didFinishSelector
```

```
-(void) setRequestMethod:(NSString*)httpMethod
-(void) startAsynchronous
-(void) startSynchronous
-(void) cancel
-(void) setUploadProgressDelegate
-(void) setDownloadProgressDelegate
-(void) setShowAccurateProgress
-(void)setMaxConcurrentHTTPRequestCount:(const unsigned char)cnt
-(NSInteger) getMaxConcurrentHTTPRequestCount
-(NSString*) responseString
-(NSData*) responseData
-addRequestHeader:(NSString*)header value:(NSString*)value
-appendPostData:(NSData*)postData
-(void) buildPostBody
-(void) setClientCertificateIdentity:(SecIdentityRef)anIdentity
-(void) setEtag:(NSString*)etag withMatchType:
(EtagMatchType)matchType;
```

Technical Details

The SDMConnectivity library wraps internally the socket based CFNetwork APIs and uses NSOperationQueue to collect and fire asynchronous requests. The number of maximum concurrent HTTP requests is limited to MAX_CONCURRENT_THREADS (Default: 5).

The `SDMHttpRequestDelegate` protocol defines default delegate methods for request status related housekeeping. Client classes can implement this protocol to hook in for requestStarted / requestFinished / requestFailed default delegates.

In cases when clients prefer to use custom selectors for request notifications, they do not need to adapt the `SDMHttpRequestDelegate` protocol, but rather register themselves as delegates and set their own selectors as didStartSelector / didFinishSelector / didFailSelector.

The `SDMProgressDelegate` defines default delegate methods for upload and download progress notification. The protocol has to be adapted by client classes to hook in for didReceiveBytes / didSendBytes / incrementDownloadSizeBy / incrementUploadSizeBy delegates. You can choose to register a download or upload progress delegate using `SDMHttpRequest` instance methods `-setUploadProgressDelegate` and `-setDownloadProgressDelegate`.

The factory method instantiates `SUPRequest` by default. SUPRequest must be used to communicate through Online Data Proxy channel; it is part of the SAP Mobile Platform ODP libraries, which have to be linked to the project. The SDMConnectivity library provides the means to transparently choose between SUPRequest using connections through the Online Data Proxy (ODP) Channel or `SDMHttpRequest` which leverages the usage of classical HTTP/HTTPS connections.

Protocols:

```
SDMHttpRequestDelegate

+ requestStarted:
+ requestFinished:
```

```
+ requestFailed:
+ requestRedirected:
+ request:didRecieveData:
+ authenticationNeededForRequest:
+ proxyAuthenticationNeededForRequest:


SDMProgressDelegate

+ setProgress:
+ request:didReceiveBytes:
+ request:didSendBytes:
+ request:incrementDownloadSizeBy:
+ request:incrementUploadSizeBy:
```

ETag Support

The SDMConnectivity library provides APIs that allow ETags to be forwarded as request headers.

HTTP header fields are components of the message header for requests and responses. These fields define the the operating parameters of an HTTP transaction. The following header fields are applicable for ETag support.

Table 1. ETag Header Fields

Header Fields	Description
If-Match	A client that has one or more ETags obtained from a resource, can verify that one of these ETags is current.
If-None-Match	A client that has one or more ETags obtained from a resource can verify that none of these ETags is current.
If-Range	A client that has an ETag that matches the current ETag of a resource, can request for the specified sub-range of the resource. This means that the client has a partial copy of an entity in its cache and can request to get the entire entity.

Note: If the backend system corresponds to Gateway, only the If-Match header field is supported.

Using ETags for HTTP Request Methods

ETags are forwarded as headers in HTTP request methods. Here are some examples for ETags with some of the commonly used request methods.

- HTTP GET
  This method requests a representation of the specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the value of the

specified resource on the server is returned. If it fails, the status code returned is 412 (Precondition Failed)

- HTTP PUT

  This method uploads the representation of the specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the specified resource is uploaded at the server. If it fails, the status code returned is 412 (Precondition Failed).
- HTTP DELETE

  This method deletes a specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the resource is deleted. If it fails, the status code returned is 412 (Precondition Failed).

Examples for Supported Features

Example - Request Initialization

```
NSString* serverUrlWithLanguage = [NSString stringWithFormat:@"%@?
sap-language=%@",[ConnectivitySettings url],[[ConnectivitySettings
instance] currentLanguage]];
[self setRequest:[SDMHTTPRequest requestWithURL:[NSURL
URLWithString: serverUrlWithLanguage]]];

NSString* serverUrlWithLanguage = [NSString stringWithFormat:@"%@?
sap-language=%@",[ConnectivitySettings url],[[ConnectivitySettings
instance] currentLanguage]];
m_AsynchRequest = [HTTPRequest requestWithURL:[NSURL URLWithString:
serverUrlWithLanguage]];

[m_AsynchRequest
setDidFinishSelector:@selector(serviceDocFetchComplete:)];

[m_AsynchRequest
setDidFailSelector:@selector(serviceDocFetchFailed:)];

[m_AsynchRequest setRequestMethod:@”POST”];
```

Example - Request Execution

```
[m_AsynchRequest startSynchronous];

[m_AsynchRequest cancel];
```

Example - Progress Tracking

```
[m_AsynchRequest setUploadProgressDelegate:m_ProgressIndicator];
```

Example - Request Payload

```
NSString* stringPayload = [m_AsynchRequest responseString];

NSData* binaryPayload = [m_AsynchRequest responseData];
```

Example - Request Setup

```
[m_AsynchRequest addRequestHeader:@"myApplicationId" value:kAppId];
[m_AsynchRequest addRequestHeader:@"deviceType" value:@"iphone"];

[m_AsynchRequest appendPostData:urlEncData];
NSData* encodedPostData = [encodedPostStr
dataUsingEncoding:NSUTF8StringEncoding];
[m_AsynchRequest appendPostData:encodedPostData];
// once we have them all, build the POST body
 [m_AsynchRequest buildPostBody];

[m_AsynchRequest addRequestHeader:@"myApplicationId" value:kAppId];
[m_AsynchRequest addRequestHeader:@"deviceType" value:@"iphone"];
```

Example - ETag Support

```
id<SDMRequesting> request;
request = [SDMRequestBuilder requestWithURL:[NSURL
URLWithString:@"http://vmw3811.wdf.sap.corp:50075/sap/opu/sdata/
sap/FINCUSTFACTSHEET/
GOSNoteCollection(CustomerNo='MG001',CompanyCode='0001',NoteID='FOL
33000000000004RAW37000000000007')"]];
[request setEtag:@"201201130306299" withMatchType:EtagIfMatch];
[request setUsername:@"user"];
[request setPassword:@"password"];
 [request setDelegate:self];
 [request
setDidFinishSelector:@selector(collectionEntriesSuccess:)];
 [request setDidFailSelector:@selector(collectionEntriesFailure:)];
 [request startAsynchronous];
```

SDMSupportability

The SDMSupportability library provides functionality for logging, tracing and performance measurement.

SDMLogger

The SDMLogger is a programming interface that provides event logging facilities. Its APIs allow generating, retrieving and displaying log items for a specific application.

List of Features

- Easy-to-use APIs
- Low level system log access (via ASL methods)
- Convenience macros which automatically add additional information such as the invoker's file and method name, and line#
- Support for all iOS log levels
- Built-in log viewer

- Export capability via e-mail from the built-in log viewer; users can choose which entries to send, and the e-mail attachment is compressed
- Globalized resource bundle (to be included by clients): contains all the various labels and accessibility hints belonging to the LogViewer, translated to English, German, French, Spanish, Portuguese, Japanese, Russian, and traditional Chinese

Logger Macros

The SDMLOG macros wrap the logger APIs and automatically enhance the log entry with information such as FILE, FUNCTION and LINE#. The following macros are available (matching the exposed APIs):

```
SDMLOGEMERGENCY(msg)
SDMLOGALERT(msg)
SDMLOGCRITICAL(msg)
SDMLOGERROR(msg)
SDMLOGWARNING(msg)
SDMLOGNOTICE(msg)
SDMLOGINFO(msg)
SDMLOGDEBUG(msg)
```

SDMLogger Public APIs

```
+ (void) enableLogging
+ (void) disableLogging
-(void) displayLogsWithLevel:(LoggingLevels)level_in
-(void) displayLogsWithLevel:(LoggingLevels)level_in
forQueryOperation:(QueryOperations)queryOperation;
-(NSArray*) retrieveLogsWithLevel:(LoggingLevels)level_in;
-(NSArray*) retrieveLogsWithLevel:(LoggingLevels)level_in
forQueryOperation:(QueryOperations)queryOperation;
-(void) logMessage:(NSString*) message_in withLevel:
(LoggingLevels)level_in andInfo:(NSString*) info_in
-(void) logEmergency:(NSString*) message_in withInfo:(NSString*)
info_in
-(void) logAlert:(NSString*) message_in withInfo:(NSString*) info_in
-(void) logCritical:(NSString*) message_in withInfo:(NSString*)
info_in
-(void) logError:(NSString*) message_in withInfo:(NSString*) info_in
-(void) logWarning:(NSString*) message_in withInfo:(NSString*)
info_in
-(void) logNotice:(NSString*) message_in withInfo:(NSString*)
info_in
-(void) logInfo:(NSString*) message_in withInfo:(NSString*) info_in
-(void) logDebug:(NSString*) message_in withInfo:(NSString*) info_in
```

Technical Details

You can enable/disable logging, and display, retrieve and generate logs.

Higher priority log messages are mapped to lower values Mac OS X / iOS system wide (see `asl.h`). Therefore, use Less or LessEqual query operation to display more critical logs. For example, in order to retrieve all log messages including the lowest, Debug level ones, use the following approach:

```
[[SDMLogger instance] displayLogsWithLevel:DebugLoggingLevel
forQueryOperation:LessEqual];
```

When generating logs, consider using the SDMLOGxxx macros, because they automatically enhance the log entry with information such as FILE, FUNCTION and LINE#.

The common methods are defined by the `SDMLogging` protocol. (See also: `SDMLogger` default implementation.)

The built-in crash log support provided by Apple can be used additionally for supportability purposes. You can retrieve crash logs either by using iTunes or by using the iPhone Configuration Utility. Note that Apple imposes restrictions on an application transmitting any data about the user without the user's prior permission, as described in the App Store review guidelines at *http://developer.apple.com/appstore/resources/approval/guidelines.html*, see Chapter 17, Privacy.

SDMPerfTimer

The SDMPerfTimer is a high precision timer class, which uses a high performance timer providing a nanosecond granularity. This timer class is for accurate performance measurements.

List of Features

- Easy-to-use, tiny API set
- Provides high precision timer data
- Low initialization overhead

SDMPerfTimer Public APIs

```
- (uint_64t) getTimeElapsedInMilisec
- (void) start
- (void) reset
```

SAP Passport

For the Single Activity Trace an SAP Passport has to be issued by the connectivity layer of the library.

The SAP Passport is transported as an HTTP header in the request. The server handles the SAP Passport to generate end-to-end Trace.

ODP SDK API Usage

The ODP SDK consists of APIs used to customize applications to send and receive data using Online Data Proxy.

For a comprehensive list of API references, extract the contents from the following zip files:

- *SMP_HOME*`\MobileSDK<Version>\OData\iOS\docs\SUPProxyClient-API-Docs.zip`
- *SMP_HOME*`\MobileSDK<Version>\OData\iOS\docs\SDMConnectivity-API-Docs.zip`
- *SMP_HOME*`\MobileSDK<Version>\OData\iOS\docs\SDMParser-API-Docs.zip`
- *SMP_HOME*`\MobileSDK<Version>\OData\iOS\docs\SDMSupportability-API-Docs.zip`

Afaria APIs

Use the Afaria APIs to provision your SAP Mobile Platform application with configuration data for connecting to the SAP Mobile Server, and certificates.

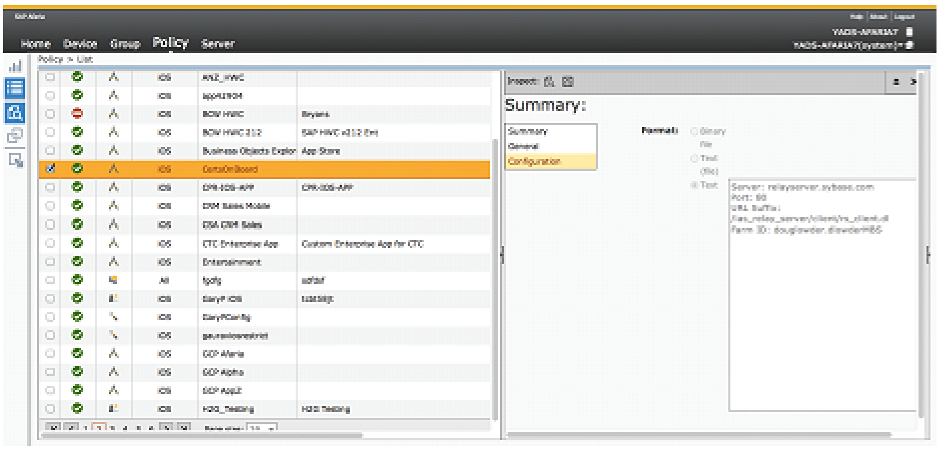
Using Afaria to Provision Configuration Data

You can use Afaria to provision configuration data for a SAP Mobile Platform application, including the SAP Mobile Server server name, port number, and other parameters.

To use these APIs you must provide the application to the device through an Afaria application policy. When setting up such an application policy, the Afaria administration interface provides an option to add configuration data to the policy as text or binary.

The following is an an example of the Afaria administration screen for an application policy that provides an application named "CertsOnBoard" to an enrolled device. The "Configuration" tab shows the configuration data provided to the application.

In this case, the configuration information is added using the administration user interface, but it can also be provided as a text or binary file. The example shows plain text, but you can also provide the information as XML or JSON text for easier parsing by the application.



iOS	ANZ_HWC	
iOS	app42904	
iOS	BCW HWC	Bryans
iOS	BCW HWC 212	SAP HWC v212 Ent
iOS	Business Objects Explor	App Store
iOS	CertsOnBoard	
iOS	CPR-IOS-APP	CPR-IOS-APP
iOS	CRM Sales Mobile	
iOS	CSA CRM Sales	
iOS	CTC Enterprise App	Custom Enterprise App for CTC
iOS	Entertainment	
All	fgdfg	sdfdsf
iOS	GaryP iOS	[illegible]
iOS	GaryPConfig	
iOS	gauravioswrestrict	
iOS	GCP Afaria	
iOS	GCP Alpha	
iOS	GCP App2	
iOS	H2G_Testing	H2G Testing

```
Server: relayserver.sybase.com
Port: 80
URL Suffix:
/ias_relay_server/client/rs_client.dll
Farm ID: douglowder.dlowderMBS
```

You can obtain configuration data for your application using Afaria by calling the following API from the `SeedingAPISynchronous` class (in Afaria's `SeedingAPISynchronous.h` header file:

```
+ (NSInteger)retrieveSeedData:(NSString *)urlScheme InFile:
(NSMutableString *)seedFile withCredentials:(NSURLCredential
*)credentials;
```

Or, call this asynchronous API from the `SeedDataAPI` class (in `SeedDataAPI.h`):

```
- (void)retrieveSeedData;
```

To access this data, the application provides an `NSMutableString` to the `retrieveSeedData` API. If the device is correctly enrolled to Afaria, the API returns `kSeedDataAvailable` and the `NSMutableString` contains the full path to a file in the application's sandbox with the seed data.

This example code retrieves the configuration data using the Afaria API, parses it using the native iOS APIs, and applies the appropriate settings using the SAP Mobile Platform APIs (the `SUPApplication` and `SUPConnectionProperties` classes).

```
NSMutableString *seedFile = [NSMutableString string];
retCode = [SeedingAPISynchronous retrieveSeedData:@"certsonboard-
seed" InFile:seedFile withCredentials:nil];
NSError *error = nil;
switch(retCode)
{
  case kSeedDataAvailable:  // Seed data is available, read the file
    NSLog(@"Seed file = %@",seedFile);
    NSLog(@"Seed data = %@",[NSString
stringWithContentsOfFile:seedFile encoding:NSUTF8StringEncoding
error:&error]);
    break;
  case kSeedDataUnavailable:
    NSLog(@"kSeedDataUnavailable"); // Error
    break;
  case kAfariaClientNotInstalled:
    NSLog(@"kAfariaClientNotInstalled");  // Error
    break;
  case kAfariaSettingsRequested:
    NSLog(@"kAfariaSettingsRequested");  // Error
    break;
}

// Read the text from the Afaria configuration file
NSString *configurationText = [NSString
stringWithContentsOfFile:seedFile encoding:NSUTF8StringEncoding
error:&error];

// Separate the text into lines
NSArray *configurationLines = [configurationText
componentsSeparatedByString:@"\n"];

// Create a dictionary, and go through the lines to find name value
pairs
```

```
NSMutableDictionary *settings = [NSMutableDictionary dictionary];
for(NSString *s in configurationLines)
{
  NSArray *nvpair = [s componentsSeparatedByString:@": "];
  if([nvpair count] == 2)
    [settings setValue:[nvpair objectAtIndex:1] forKey:[nvpair
objectAtIndex:0]];
}

// Use the name value pairs from the configuration file to set the
appropriate settings in the SUPApplication API
SUPApplication *app = [SUPApplication getInstance];
app.applicationIdentifier = @"myAppID";

SUPConnectionProperties *properties = app.connectionProperties;

properties.serverName = [settings valueForKey:@"Server"];
properties.portNumber = [[settings valueForKey:@"Port"] intValue];
properties.farmId = [settings valueForKey:@"Farm ID"];
properties.urlSuffix = [settings valueForKey:@"URL Suffix"];

NSLog(@"Server name is set to %@",properties.serverName);
NSLog(@"Port number is set to %d",properties.portNumber);
NSLog(@"Farm ID is set to %@",properties.farmId);
NSLog(@"URL suffix is set to %@",properties.urlSuffix);
```

Example output on the Xcode console:

```
2012-09-24 13:06:33.014 CertsOnboard[579:707] Seed file = /var/
mobile/Applications/21935FE8-843A-418D-A2BF-EE415B5D4DF0/Documents/
TEXT_FILE_
2012-09-24 13:06:33.016 CertsOnboard[579:707] Seed data = Server:
relayserver.sybase.com

Port: 80
URL Suffix: /ias_relay_server/client/rs_client.dl
Farm ID: example.exampleMBS
```

For more information on the Afaria APIs and the meanings of return codes, see the Afaria documentation.

Using Certificates from Afaria for Authentication

One of the features of Afaria is the ability to provide a device with a signed certificate that could be used as an authentication credential for SAP Mobile Platform. This note explains how to take a certificate provided by Afaria and convert it into a form suitable for use with SAP Mobile Platform.

Prerequisites:

- The iOS application has been built using the SAP Mobile Platform generated code and framework headers and libraries.
- The iOS application includes the required Afaria headers `SeedDataAPI.h` and `SeedingAPISynchronous.h`.

- The iOS application has been registered with the Afaria server as an application policy and made available to the iOS client device.

In SAP Mobile Platform, a certificate can be used for authentication by creating a LoginCertificate object (the SUPLoginCertificate class), and setting that as the certificate property in the client's synchronization profile. The login certificate has two properties that are used in authentication; the subjectCN (the common name of the certificate) and the signedCertificate (the certificate data itself).

After calling the Afaria APIs to get initial settings and configuration data, an application using Afaria may obtain a signed certificate using one of these APIs:

```
+ (NSInteger)retrieveCertificateWithPrivateKey:
(SecKeyRef)privateKey andPublicKey:(SecKeyRef)publicKey
andCommonName:(NSString *)commonName andChallenge:(NSString
*)challengeCode forUrlScheme:(NSString *)urlScheme inCertificate:
(SecCertificateRef *)certificate;


+ (NSInteger)retrieveCertificateWithUrl:(NSURL *)url andPrivateKey:
(SecKeyRef)privateKey andPublicKey:(SecKeyRef)publicKey
andCommonName:(NSString *)commonName andChallenge:(NSString
*)challengeCode inCertificate:(SecCertificateRef *)certificate;
```

After this, the application will have a SecCertificateRef with the certificate, and a SecKeyRef with the private key. The certificate data in the SecCertificateRef cannot be used as is in the signedCertificate property of an SUPLoginCertificate. The signedCertificate property value is expected to contain the certificate and a digest of the certificate in ASN.1 format. To create the signedCertificate property value:

This sample code shows how to get the Afaria certificate, create an `SUPLoginCertificate` object, and attach it to a SAP Mobile Platform synchronization profile.

```
// At this point, an Afaria user should have a signed certificate and
a private key available after importing
// their certificate using either of the Afaria APIs
 /*

+ (NSInteger)retrieveCertificateWithPrivateKey:
(SecKeyRef)privateKey andPublicKey:(SecKeyRef)publicKey
andCommonName:(NSString *)commonName andChallenge:(NSString
*)challengeCode forUrlScheme:(NSString *)urlScheme inCertificate:
(SecCertificateRef *)certificate;

+ (NSInteger)retrieveCertificateWithUrl:(NSURL *)url andPrivateKey:
(SecKeyRef)privateKey andPublicKey:(SecKeyRef)publicKey
andCommonName:(NSString *)commonName andChallenge:(NSString
*)challengeCode inCertificate:(SecCertificateRef *)certificate;

SecCertificateRef certificate;
SecKeyRef privatekey;

*/
```

```
SUPLoginCertificate *loginCertificate = [SUPLoginCertificate
getInstance];

loginCertificate.subjectCN =
(NSString*)SecCertificateCopySubjectSummary(certificate);

loginCertificate.signedCertificate = [CertBlobUtility
makeCertBlob:certificate andPrivateKey:privatekey];

NSLog(@"Certificate created. Subject =
%@",loginCertificate.subjectCN);

NSLog(@"MD5 digest = %@",[CertBlobUtility
md5sum:loginCertificate.signedCertificate]);

NSLog(@"SHA1 digest = %@",[CertBlobUtility
sha1:loginCertificate.signedCertificate]);


// Attach certificate to sync profile

SUPConnectionProfile *syncProfile = [SAPSSOCertTestSAPSSOCertTestDB
getSynchronizationProfile];
syncProfile.certificate = loginCertificate;
[loginCertificate release];
```

CertBlobUtility Header

The CertBlob Utility header of the CertBlob class.

```
#import <Foundation/Foundation.h>
#import <Security/Security.h>

@interface CertBlobUtility : NSObject

// Returns the MD5 sum of the input data
+ (NSString*)md5sum:(NSData*)certData;

// Returns the SHA1 fingerprint of the input data
+ (NSString*)sha1:(NSData*)certData;

// Given a signed certificate and private key, return a certificate
blob suitable for use in an SUPLoginCertificate
+ (NSData *)makeCertBlob:(SecCertificateRef)certificate
andPrivateKey:(SecKeyRef)privateKey;
@end
```

CertBlobUtility Source

The CertBlob Utility source of the CertBlob class.

```
#import "CertBlobUtility.h"
#import <CommonCrypto/CommonDigest.h>


bool getAsn1LengthBytes(
  int iLengthVal,  // (IN) value to be encoded
  unsigned char* pbOut,     // (IN/OUT) buffer to be populated with
the encoding or NULL to get sizing information
  int *iOutLen    // (IN/OUT) if pbOut != NULL, size of pbOut buffer
in allocated bytes.  Is set to the number
  // of bytes required/written in the encoding on return.
);

bool makeCertBlob(
  unsigned char* pbCert, // Certificate to be encoded in the CertBlob
  int iCertLen, // Length in bytes of pbCert
  unsigned char* pbSig,  // Signature to be encoded in the CertBlob
  int iSigLen,  // Length in bytes of pbSig
  unsigned char byteAlgorithm, // Algorithm constant to be encoded in
the CertBlob
  unsigned char* pbOut, // (IN/OUT) buffer to be populated with the
encoding or NULL to get sizing information
  int *iOutLen // (IN/OUT) if pbOut != NULL, size of pbOut buffer in
allocated bytes.  Is set to the number
  // of bytes required/written in the encoding on return.
);


bool getAsn1LengthBytes(
  int iLengthVal,  // (IN) value to be encoded
  unsigned char* pbOut,     // (IN/OUT) buffer to be populated with
the encoding or NULL to get sizing information
  int *iOutLen    // (IN/OUT) if pbOut != NULL, size of pbOut buffer
in allocated bytes.  Is set to the number
  // of bytes required/written in the encoding on return.
)
{
  // simple short form length
  if ( iLengthVal < 0x80 )
  {
    if ( ( pbOut != NULL ) && ( *iOutLen < 1 ) )
      return false;

    *iOutLen = 1;
    if ( pbOut != NULL )
      *pbOut = (unsigned char) iLengthVal;
    return true;
  }

  // if we got here, we need long form, because the short form doesn't
fit in a single byte

  // count the number of bytes in iVal
  int iTmp = iLengthVal;
  int iCount = 0;
  iTmp = iLengthVal;
  unsigned char byteLast = 0;
  while ( iTmp != 0 )
  {
    iCount++;
    byteLast = (unsigned char) ( iTmp & 0xFF );
    iTmp >>= 8;
  }

  // case where caller wants to know how to size buffer
  if ( NULL == pbOut )
  {
    *iOutLen = iCount + 1; // +1 for the length byte
      return true;
  }

  if ( *iOutLen < iCount + 1 )
    return false;

  *iOutLen = iCount + 1; // +1 for the length byte

  // Create an array with the count of bytes, followed by the iVal
bytes
  // Setting the top bit of the count indicates that this is a count
with the value to follow, not the actual integer value
  pbOut[ 0 ] = (unsigned char) ( iCount | 0x80 );  // count
  iTmp = iLengthVal;
  while ( iTmp != 0 )
  {
    unsigned char b = (unsigned char) ( iTmp & 0xFF );
    iTmp >>= 8;
    pbOut[ iCount-- ] = b;
  }

  return true;
}

// makeCertBlob "C" function used by SSOCertManager makeCertBlob
method below
/*
 * Returns a buffer containing an ASN.1 encoding for a CertBlob.
 * Upon return, pbOut will be filled with the result and
 * iOutLen will contain the number of bytes written.  If this
 * function is called with NULL as the pbOut pointer, it will
 * populate iOutLen without writing anything.  The expected usage
 * is to call with pbOut==NULL to size the buffer, allocate the
buffer,
 * then call it again with the newly allocated buffer.
 *
 * Return value of false is if pbOut!=NULL and the passed in iOutLen
 * is less than the required number of bytes to write the result.
 *
 */
bool makeCertBlob(
  unsigned char* pbCert, // Certificate to be encoded in the CertBlob
  int iCertLen, // Length in bytes of pbCert
  unsigned char* pbSig,  // Signature to be encoded in the CertBlob
  int iSigLen,  // Length in bytes of pbSig
  unsigned char byteAlgorithm, // Algorithm constant to be encoded in
the CertBlob
  unsigned char* pbOut, // (IN/OUT) buffer to be populated with the
encoding or NULL to get sizing information
```

```
  int *iOutLen // (IN/OUT) if pbOut != NULL, size of pbOut buffer in
allocated bytes.  Is set to the number
  // of bytes required/written in the encoding on return.
  )
{
  int iCertLenLen, iSigLenLen;
  int iAlgorithmLen = 2;

  // get number of bytes in length descriptors
  if ( !getAsn1LengthBytes( iCertLen, NULL, &iCertLenLen ) )
    return false;

  if ( !getAsn1LengthBytes( iSigLen, NULL, &iSigLenLen ) )
    return false;

  // calculate size of content of sequence
  int iSeqLen = 1 +  // type code for OCTET STRING
  iCertLenLen +   // length bytes for Certificate
  iCertLen +      // data bytes for Certificate
  1 +             // type code for OCTET STRING
  iSigLenLen +    // length bytes for Signature
  iSigLen +       // data bytes for Signature
  1 +             // type code for INTEGER
  iAlgorithmLen;  // data bytes for algorithm (assumed to be an
integer that fits in a single byte)

  // now calculate size of outer sequence
  int iSeqLenLen;
  if ( !getAsn1LengthBytes( iSeqLen, NULL, &iSeqLenLen ) )
    return false;

  int iTotalLen = 1 +           // type code for SEQUENCE
  iSeqLenLen +  // length bytes for Sequence
  iSeqLen;      // data bytes for Sequence

  if ( NULL == pbOut )
  {
    // caller is just asking for required buffer size
    *iOutLen = iTotalLen;
      return true;
  }

  // test whether buffer is large enough
  if ( *iOutLen < iTotalLen )
    return false;

  // write everything to the buffer
  int iCurIdx = 0;

  // header bytes for wrapping sequence
  pbOut[ iCurIdx++ ] = (unsigned char) 0x30;  // type code for
SEQUENCE
  if ( !getAsn1LengthBytes( iSeqLen, pbOut + iCurIdx,
&iSeqLenLen ) )   // length bytes for Sequence
    return false;
  iCurIdx += iSeqLenLen;
  // first element of sequence -> certificate
  pbOut[ iCurIdx++ ] = (unsigned char) 0x04;  // type code for OCTET
STRING
  if ( !getAsn1LengthBytes( iCertLen, pbOut + iCurIdx,
&iCertLenLen ) )  // length bytes for Certificate
    return false;
  iCurIdx += iCertLenLen;
  memcpy( pbOut + iCurIdx, pbCert, iCertLen );  // bytes for
Certificate
  iCurIdx += iCertLen;

  // second element of sequence -> signature
  pbOut[ iCurIdx++ ] = (unsigned char) 0x04;  // type code for OCTET
STRING
  if ( !getAsn1LengthBytes( iSigLen, pbOut + iCurIdx,
&iSigLenLen ) )  // length bytes for Certificate
    return false;

  iCurIdx += iSigLenLen;
  memcpy( pbOut + iCurIdx, pbSig, iSigLen );  // bytes for
Certificate
  iCurIdx += iSigLen;

  // third element of sequence -> algorithm
  pbOut[ iCurIdx++ ] = (unsigned char) 0x02;  // type code for INTEGER
  pbOut[ iCurIdx++ ] = (unsigned char) 0x01;  // length bytes for
value (assume 1)
  pbOut[ iCurIdx++ ] = byteAlgorithm;  // algorithm constant

    return true;
}

@implementation CertBlobUtility

+ (NSString*)md5sum:(NSData*)certData
{

  CC_MD5_CTX md5;

  CC_MD5_Init(&md5);

  CC_MD5_Update(&md5, [certData bytes], [certData length]);

    unsigned char digest[CC_MD5_DIGEST_LENGTH];

    CC_MD5_Final(digest, &md5);

    NSString* s = [NSString stringWithFormat: @"%02x%02x%02x%02x%02x
%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x",
      digest[0], digest[1],
      digest[2], digest[3],
      digest[4], digest[5],
      digest[6], digest[7],
      digest[8], digest[9],
      digest[10], digest[11],
      digest[12], digest[13],
      digest[14], digest[15]];
    return s;
}

+ (NSString*)sha1:(NSData*)certData {
  unsigned char sha1Buffer[CC_SHA1_DIGEST_LENGTH];
  CC_SHA1(certData.bytes, certData.length, sha1Buffer);
  NSMutableString *fingerprint = [NSMutableString
stringWithCapacity:CC_SHA1_DIGEST_LENGTH * 3];
  for (int i = 0; i < CC_SHA1_DIGEST_LENGTH; ++i)
    [fingerprint appendFormat:@"%02x ",sha1Buffer[i]];
  return [fingerprint stringByTrimmingCharactersInSet:
[NSCharacterSet whitespaceCharacterSet]];
}

// SSOCertManager makeCertBlob: used by getCertBlob: API below
// Makes a certBlob from given certificate and private key and
returns it
+ (NSData *)makeCertBlob:(SecCertificateRef)certificate
andPrivateKey:(SecKeyRef)privateKey {
    NSData *sigData;
    NSData *certData;

    CFDataRef certCFData = SecCertificateCopyData(certificate);
    unsigned char certDigest[CC_SHA1_DIGEST_LENGTH];
    CC_SHA1( CFDataGetBytePtr(certCFData),
CFDataGetLength(certCFData), certDigest );

    certData = [NSData dataWithBytes:CFDataGetBytePtr(certCFData)
length:CFDataGetLength(certCFData)];

    size_t sigLen = 1024;
    uint8_t sigBuf[sigLen];

    // Encrypt the digest of the certificate with private key
    OSStatus err = SecKeyRawSign(privateKey, kSecPaddingPKCS1,
                                 certDigest,
CC_SHA1_DIGEST_LENGTH, //data.bytes, data.length,
                                 sigBuf, &sigLen);

    if (err == noErr) {
        sigData = [NSData dataWithBytes:sigBuf length:sigLen];
    }
    if ( certCFData != NULL )
        CFRelease(certCFData);

    if ( ( certData == nil ) || ( sigData == nil ) )
        return nil;

    int iLength = 0;
    if ( ( !makeCertBlob( (unsigned char *)[certData bytes],
[certData length], (unsigned char *)[sigData bytes], [sigData
length], 1, NULL, &iLength ) ) || ( iLength == 0 ) )
        return nil;
    unsigned char* pBuf = (unsigned char*)malloc(iLength);
    if ( !makeCertBlob( (unsigned char *)[certData bytes], [certData
length], (unsigned char *)[sigData bytes], [sigData length], 1, pBuf,
&iLength ) ) {
        free( pBuf );
        return nil;
    }

    NSData* certBlob = [NSData dataWithBytes:pBuf length:iLength];
    free( pBuf );

    return certBlob;
}


@end
```

Security APIs

The security APIs allow you to customize some aspects of secure storage.

DataVault

The `DataVault` class provides encrypted storage of occasionally used, small pieces of data. All exceptions thrown by `DataVault` methods are of type `DataVaultException`.

By linking the `libDatavault.a` static library, you can use the `DataVault` class for on-device persistent storage of certificates, database encryption keys, passwords, and other sensitive items. Use this class to:

- Create a vault
- Set a vault's properties
- Store objects in a vault
- Retrieve objects from a vault
- Change the password used to access a vault
- Control access for a vault that is shared by multiple iOS application.

The contents of the data vault are strongly encrypted using AES-256. The `DataVault` class allows you create a named vault, and specify a password and salt used to unlock it. The password can be of arbitrary length and can include any characters. The password and salt together generate the AES key. If the user enters the same password when unlocking, the contents are decrypted. If the user enters an incorrect password, exceptions occur. If the user enters an incorrect password a configurable number of times, the vault is deleted and any data stored within it becomes unrecoverable. The vault can also relock itself after a configurable amount of time.

Typical usage of the `DataVault` is to implement an application login screen. Upon application start, the user is prompted for a password, which unlocks the vault. If the unlock attempt is successful, the user is allowed into the rest of the application. User credentials for synchronization can also be extracted from the vault so the user need not reenter passwords.

createVault

Creates a new secure store (a vault)

A unique name is assigned, and after creation, the vault is referenced and accessed by that name. This method also assigns a password and salt value to the vault. If a vault with the same name already exists, this method throws an exception. A newly created vault is in the unlocked state.

Syntax

```
+ (DataVault*)createVault:(NSString*)name Password:
(NSString*)password Salt:(NSString*)salt;
```

Parameters

- **name –** an arbitrary name for a `DataVault` instance on this device. This name is effectively the primary key for looking up `DataVault` instances on the device, so it cannot use the same name as any existing instance. If it does, this method throws an exception with error code INVALID_ARG. The name also cannot be empty or null.
- **password –** the initial encryption password for this `DataVault`. This is the password needed for unlocking the vault. If null is passed, a default password is computed and used.
- **salt –** the encryption salt value for this `DataVault`. This value, combined with the password, creates the actual encryption key that protects the data in the vault. If null is passed, a default salt is computed and used.

Returns

Returns the newly created instance of the `DataVault` with the provided ID. The returned `DataVault` is in the unlocked state with default configuration values. To change the default configuration values, you can immediately call the "set" methods for the values you want to change.

Examples

- **Create a data vault –** creates a new data vault called `myVault`.

  ```
  @try
  {
      if(![DataVault vaultExists:@"myVault"])
      {
          oVault = [DataVault createVault:@"myVault"
                                 Password:@"goodPassword"
                                 Salt:@"goodSalt"];
      }
  }
  @catch ( NSException *e )
  {
  ```

```
    NSLog(@"DataVaultException: %@",[e description]);
}
```

vaultExists

Tests whether the specified vault exists.

Syntax

```
+ (BOOL)vaultExists:(NSString*)name;
```

Parameters

- **name –** the vault name.

Returns

Returns true if the vault exists; otherwise returns false.

Examples

- **Check if a data vault exists –** checks if a data vault called `myVault` exists, and if so, deletes it.

```
if ([DataVault vaultExists:@"myVault"])
{
    [DataVault deleteVault:@"myVault"];
}
```

getVault

Retrieves a vault.

Syntax

```
+ (DataVault*)getVault:(NSString*)name;
```

Parameters

- **name –** the vault name.

Returns

Returns a `DataVault` instance.

If the vault does not exist, a `DataVaultException` is thrown.

deleteVault

Deletes the specified vault from on-device storage.

Note: When you have a shared data vault, if one application deletes the vault, the data vault is no longer accessible in the other application.

If the vault does not exist, this method throws an exception. The vault need not be in the unlocked state, and can be deleted even if the password is unknown.

Syntax

```
+ (void)deleteVault:(NSString*)name;
```

Parameters

- **name –** the vault name.

Examples

- **Delete a data vault –** deletes a data vault called `myVault`.

```
@try
{
    if([DataVault vaultExists:@"myVault"])
    {
        [DataVault deleteVault:@"myVault"];
    }
}
@catch ( NSException *e )
{
    NSLog(@"DataVaultException: %@",[e description]);
}
```

getDataNames

Retrieves information about the data names stored in the vault.

The application can pass the data names to `getValue` or `getString` to retrieve the data values.

Syntax

```
- (NSArray *)getDataNames;
```

Parameters

None.

Returns

Returns a list of objects of type `DVDataName`.

Examples

- **Get data names**

```
// Call getDataNames to retrieve all stored element names from our
data vault
NSArray *dataNames = [dataVault getDataNames];
if (dataNames != nil) {
  DVDataName *dataName;
  for (NSInteger iIdx = 0; iIdx < [dataNames count]; iIdx++) {
    dataName = [dataNames objectAtIndex:iIdx];
    if (dataName.type == kDVDataTypeString) {
      // Stored value is of string type
      NSString *thisStringValue = [dataVault
getString:dataName.name];
    }
    else if (dataName.type == kDVDataTypeBinary) {
      // Stored value is of binary type
      NSData *thisBinaryValue = [dataVault
getValue:dataName.name];
    }
    else {
      // Unknown type. Possibly stored using previous version of
dataVault
      // Try as string first and then as binary
      NSString *thisStringValue = [dataVault
getString:dataName.name];
      if (thisStringValue == nil) {
        NSData *thisBinaryValue = [dataVault
getValue:dataName.name];
      }
    }
  }
}
```

setPasswordPolicy

Stores the password policy and applies it when `changePassword` is called, or when validating the password in the `unlock` method.

If the application has not set a password policy using this method, the data vault does not validate the password in the `createVault` or `changePassword` method. An exception is thrown if there is any invalid (negative) value in the `passwordPolicy` object.

Syntax

```
- (void)setPasswordPolicy:DVPasswordPolicy oPasswordPolicy;
```

Parameters

- **oPasswordPolicy –** the password policy constraints.

Returns

None.

Examples

- **Set a password policy**

```
// setPasswordPolicy locks the vault to ensure the old password
conforms to the new password policy settings
[dataVault setPasswordPolicy:pwdPolicy];
```

Password Policy Structure

A structure defines the policy used to generate the password.

Table 2. Password Policy Structure

Name	Type	Description
defaultPasswordAllowed	Boolean	Indicates if client application is allowed to use default password for the data Vault. If this is set to TRUE and if client application uses default password then minLength, hasDigits, hasUpper, hasLower and hasSpecial parameters in the policy are ignored.
minimumLength	Integer	The minimum length of the password.
hasDigits	Boolean	Indicates if the password must contain digits.
hasUpper	Boolean	Indicates if the password must contain uppercase characters.
hasLower	Boolean	Indicates if the password must contain lowercase characters.
hasSpecial	Boolean	Indicates if the password must contain special characters. The set of special characters is: “~!@#$%^&*()-+”.
expirationDays	Integer	Specifies password expiry days from the date of setting the password. 0 indicates no expiry.

Name	Type	Description
minUniqueChars	Integer	The minimum number of unique characters in the password. For example, if length is 5 and minUniqueChars is 4 then “aaate” or “ababa” would be invalid passwords. Instead, “aaord” would be a valid password.
lockTimeout	Integer	The timeout value (in seconds) after which the vault will be locked from the unlock time. 0 indicates no timeout. This value overrides the value set by `setLockTimeout` method.
retryLimit	Integer	The number of failed unlock attempts after which data vault is deleted. 0 indicates no retry limit. This value overrides the value set by the `setRetryLimit` method.

Settings for Password Policy

The client applications use these settings to fill the PasswordPolicy structure. The default values are used by the data vault when no policy is configured. The defaults are also used in SAP Control Center in the default template. The SAP Mobile Platform administrator can modify these settings through SAP Control Center. The application must set the password policy for the data vault with the administrative (or alternative) settings.

Note: Setting the password policy locks the vault. The password policy is enforced when `unlock` is called (because the password is not saved, calling `unlock` is the only time that the policy can be evaluated).

- **MCL_PROP_ID_PWDPOLICY_ENABLED –** Boolean property with a default value of false. Indicates if a password policy is enabled by the administrator.
- **MCL_PROP_ID_PWDPOLICY_DEFAULT_PASSWORD_ALLOWED –** Boolean property with a default value of false. Indicates if the client application is allowed to use the default password for the data vault.
- **MCL_PROP_ID_PWDPOLICY_MIN_LENGTH –** Integer property with a default value of 0. Defines the minimum length for the password.
- **MCL_PROP_ID_PWDPOLICY_HAS_DIGITS –** Boolean property with a default value of false. Indicates if the password must contain digits.

- **MCL_PROP_ID_PWDPOLICY_HAS_UPPER –** Boolean property with a default value of false. Indicates if the password must contain at least one uppercase character.
- **MCL_PROP_ID_PWDPOLICY_HAS_LOWER –** Boolean property with a default value of false. Indicates if the password must contain at least one lowercase character.
- **MCL_PROP_ID_PWDPOLICY_HAS_SPECIAL –** Boolean property with a default value of false. Indicates if the password must contain at least one special character. A special character is a character in this set "~!@#$%^&*()-+".
- **MCL_PROP_ID_PWDPOLICY_EXPIRATION_DAYS –** Integer property with a default value of 0. Specifies the number of days in which password will expire from the date of setting the password. Password expiration is checked only when the vault is unlocked.
- **MCL_PROP_ID_PWDPOLICY_MIN_UNIQUE_CHARS –** Integer property with a default value of 0. Specifies minimum number of unique characters in the password. For example, if minimum length is 5 and minUniqueChars is 4 then "aaate" or "ababa" would be invalid passwords. Instead, "aaord" would be a valid password.
- **MCL_PROP_ID_PWDPOLICY_LOCK_TIMEOUT –** Integer property with a default value of 0. Specifies timeout value (in seconds) after which the vault is locked from the unlock time. 0 indicates no timeout.
- **MCL_PROP_ID_PWDPOLICY_RETRY_LIMIT –** Integer property with a default value of 0. Specifies the number of failed unlock attempts after which data vault is deleted. 0 indicates no retry limit.

Password Errors

Password policy violations cause exceptions to be thrown.

Table 3. Password Errors

Name	Value	Description
PASSWORD_REQUIRED	50	Indicates that a blank or null password was used when the password policy does not allow default password.
PASSWORD_UNDER_MIN_LENGTH	51	Indicates that the password length is less than the required minimum.
PASSWORD_REQUIRES_DIGIT	52	Indicates that the password does not contain digits.
PASSWORD_REQUIRES_UPPER	53	Indicates that the password does not contain upper case characters.

Name	Value	Description
PASSWORD_RE-QUIRES_LOWER	54	Indicates that the password does not contain lower case characters.
PASSWORD_RE-QUIRES_SPECIAL	55	Indicates that the password does not contain one of these special characters: ~!@#$%^&*()-+.
PASSWORD_UN-DER_MIN_UNIQUE	56	Indicates that the password contains fewer than the minimum required number of unique characters.
PASSWORD_EXPIRED	57	Indicates that the password has been in use longer than the number of configured expiration days.

getPasswordPolicy

Retrieves the password policy set by `setPasswordPolicy`.

Use this method once the `DataVault` is unlocked.

Syntax

```
+ (DataVault*)getPasswordPolicy:();
```

Parameters

None.

Returns

Returns a `passwordPolicy` structure that contains the policy set by `setPasswordPolicy`.

Returns a `DVPasswordPolicy` object with the default values if no password policy is set.

Examples

- **Get the current password policy**

  ```
  // Use getPasswordPolicy to get the current policy set in the
  vault
       pwdPolicy = [dataVault getPasswordPolicy];
  ```

isDefaultPasswordUsed

Checks whether the default password is used by the vault.

Use this method once the `DataVault` is unlocked.

Syntax

```
- (BOOL)isDefaultPasswordUsed;
```

Parameters

None.

Returns

Returns	Indicates
TRUE	Both the default password and the default salt are used to encrypt the vault.
FALSE	Either the default password or the default salt are not used to encrypt the vault.

Examples

- **Check if default password used**

  ```
  // isDefaultPasswordused should return YES as nil is passed as
  password in previous step
  BOOL isDefaultPasswordUsed = [dataVault isDefaultPasswordUsed];
  ```

  This code example lacks exception handling. For a code example that includes exception handling, see *Developer Guide: OData SDK > Developing iOS Applications > ODK SDK API Usage > Security APIs > DataVault*.

lock

Locks the vault.

Once a vault is locked, you must unlock it before changing the vault's properties or storing anything in it. If the vault is already locked, `lock` has no effect.

Syntax

```
- (void)lock;
```

Parameters

None.

Returns

None.

Examples

- **Locks the data vault –** prevents changing the vaults properties or stored content.

  ```
  [oVault lock];
  ```

isLocked

Checks whether the vault is locked.

Syntax

```
- (BOOL)isLocked;
```

Parameters

None.

Returns

Re-turns	Indicates
YES	The vault is locked.
NO	The vault is unlocked.

unlock

Unlocks the vault.

Unlock the vault before changing the its properties or storing anything in it. If the incorrect password or salt is used, this method throws an exception. If the number of unsuccessful attempts exceeds the retry limit, the vault is deleted.

The password is validated against the password policy if it has been set using `setPasswordPolicy`. If the password is not compatible with the password policy, an `IncompatiblePassword` exception is thrown. In that case, call `changePassword` to set a new password that is compatible with the password policy.

Syntax

```
- (void)unlock:(NSString*)password Salt:(NSString*)salt;
```

Parameters

- **password –** the initial encryption password for this `DataVault`. If null is passed, a default password is computed and used.
- **salt –** the encryption salt value for this `DataVault`. This value, combined with the password, creates the actual encryption key that protects the data in the vault. If null is passed, a default salt is computed and used.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `kDataVaultExceptionReasonInvalidPassword`.

Examples

- **Unlocks the data vault –** once the vault is unlocked, you can change its properties and stored content.

```
@try
{
   [oVault unlock:@"password" Salt:@"salt"];
}
@catch(DataVaultException *e)
{
   NSLog(@"Exception will be thrown for bad password");
}
```

setString

Stores a string object in the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
- (void)setString:(NSString*)name Value:(NSString*)value;
```

Parameters

- **name –** the name associated with the string object to be stored.
- **value –** the string object to store in the vault.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `kDataVaultExceptionReasonInvalidPassword`.

Examples

- **Set a string value –** creates a test string, unlocks the vault, and sets a string value associated with the name `"testString"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
NSString *teststring = @"ABCDEFabcdef";
@try {
   [oVault unlock:@"goodPassword" Salt:@"goodSalt"];
   [oVault setString:@"testString" Value:teststring];
}
@catch (NSException *e) {
   NSLog(@"Exception: %@",[e description]);
}
@finally {
   [oVault lock];
}
```

getString

Retrieves a string value from the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
- (NSString*)getString:(NSString*)name;
```

Parameters

- **name –** the name associated with the string object to be retrieved.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `kDataVaultExceptionReasonInvalidPassword`.

Examples

- **Get a string value –** unlocks the vault and retrieves a string value associated with the name `"testString"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
NSString *retrievedstring = nil;

@try {
   [oVault unlock:@"goodPassword" Salt:@"goodSalt"];
   retrievedstring = [oVault getString:@"testString"];
}
@catch (NSException *e) {
   NSLog(@"Exception: %@",[e description]);
}
```

```
@finally {
   [oVault lock];
}
```

setValue

Stores a binary object in the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
- (void)setValue:(NSString*)name Value:(NSData*)value;
```

Parameters

- **name –** the name associated with the binary object to be stored.
- **value –** the binary object to store in the vault.

Returns

None.

Examples

- **Set a binary value –** unlocks the vault and stores a binary value.

```
NSString * stringTobeConverted=@"testValue";
NSData *data=[stringToBeConverted
dataUsingEncoding:NSUTF8StringEncoding];
[oVault setValue:@"testValue" Value:data];
```

getValue

Retrieves a binary object from the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
- (NSData*)getValue:(NSString*)name;
```

Parameters

- **name –** the name associated with the binary object to be retrieved.

Returns

None.

Examples

- **Get a binary value –** unlocks the vault and retrieves a binary value.

  ```
  NSData *myData=[oVault getValue:@”BinaryValue”];
  NSString * retrievedvalue=[[NSString alloc] initWithData
  encoding:NSUTF8StringEncoding];
  NSLog(@”retrieved value from the vault is %@”,retrievedvalue);
  ```

deleteValue

Deletes the specified value.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
+ (void)deleteValue:(NSString*)name;
```

Parameters

- **name –** the name of the value to be deleted.

Returns

None.

Examples

- **Delete a value –** deletes a value called `myValue`.

  ```
  [DataVault deleteValue:@"myValue"];
  ```

changePassword (two parameters)

Changes the password for the vault. Use this method when the vault is unlocked.

Modifies all name/value pairs in the vault to be encrypted with a new password/salt. If the vault is locked or the new password is empty, an exception is thrown.

Syntax

```
- (void)changePassword:(NSString*)newPassword Salt:
(NSString*)newSalt;
```

Parameters

- **newPassword –** the new password.
- **newSalt –** the new encryption salt value.

Returns

None.

Examples

- **Change the password for a data vault –** changes the password to `"newPassword"`. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
@try
{
   [oVault unlock:@"goodPassword" Salt:@"goodSalt"];
   [oVault changePassword:@"newPassword" Salt:@"newSalt"];
}
@catch (NSException *e) {
   NSLog(@"Exception: %@",[e description]);
}
@finally
{
   [oVault lock];
}
```

changePassword (four parameters)

Changes the password for the vault. Use this method when the vault is locked

This overloaded method ensures the new password is compatible with the password policy, uses the current password to unlock the vault, and changes the password of the vault to a new password. If the current password is not valid an `InvalidPassword` exception is thrown. If the new password is not compatible with the password policy set in `setPasswordPolicy` then an `IncompatiblePassword` exception is thrown.

Syntax

```
- (void)changePassword:(NSString*)currentPassword:
(NSString*)currentSalt:(NSString*)newPassword:(NSString*)newSalt;
```

Parameters

- **currentPassword –** the current encryption password for this data vault. If a null value is passed, a default password is computed and used.
- **currentSalt –** the current encryption salt value for this data vault. If a null value is passed, a default password is computed and used.
- **newPassword –** the new encryption password for this data vault. If a null value is passed, a default password is computed and used.
- **newSalt –** the new encryption salt value for this data vault. This value, combined with the password, creates the actual encryption key that protects the data in the vault. This value may be an application-specific constant. If a null value is passed, a default password is computed and used.

Returns

None.

Examples

- **Change the password for a data vault**

  ```
  // Call changePassword with four parameters, even if the vault is
  locked.
  // Pass null for oldSalt and oldPassword if the defaults were
  used.
  [dataVault changePassword:nil currentSalt:nil
  newPassword:@"password!1A" newSalt:@"saltD#ddg#k05%gnd[!1A"];
  ```

resetLockTimeout

Resets the lock timeout effectively extending the timeout period from now.

If vault is locked, a DataVaultException is thrown with the reason `kDataVaultExceptionReasonLocked`.

Syntax

```
(void)resetLockTimeout;
```

Parameters

None.

Returns

None.

Examples

- **Resets the lock timeout –** Extends the timeout period of vault

  ```
  [dataVault resetLockTimeout];
  ```

setAccessGroup

Sets the access group if multiple application share a data vault.

This method is used only for iOS applications, and must be called before accessing any `DataVault` methods. The access group must be set only if a vault is shared by multiple iPhone applications. If the vault is used only by one application, do not set the access group. The access group is listed in the `keychain-access-groups` property of the `entitlements` plist file. The recommended format is `".com.yourcompany.DataVault"`.

Syntax

```
+ (void)setAccessGroup:(NSString *)accessGroup;
```

Parameters

- **accessGroup –** The access group name.

Examples

- **Sets the Access Group Name –** Sets the access group name so that multiple iOS applications can access the data vault.

  ```
  [oVault
  setAccessGroup:@"accessGroupName.com.yourcompany.DataVault"];
  ```

Code Sample

Create a data vault for encrypted storage of application data.

```
DataVault* dataVault = nil;
@try
{
  // If the dataVault already exists, call getVault and unlock it
  // If not, create the vault with necessary password
  // The password is chosen to make sure it satisfies password policy
criteria given below
  if ( [DataVault vaultExists:@"SampleVault"] ) {
    dataVault = [DataVault getVault:@"SampleVault"];
    [dataVault unlock:@"password!1A" Salt:@"saltD#ddg#k05%gnd[!1A"];
  }
  else {
    dataVault = [DataVault createVault:@"SampleVault"
Password:@"password!1A"  Salt:@"saltD#ddg#k05%gnd[!1A"];
  }

  // Supply various criteria for password policy
  DVPasswordPolicy *pwdPolicy = [[[DVPasswordPolicy alloc] init]
autorelease];
  pwdPolicy.defaultPasswordAllowed = YES;
  pwdPolicy.minLength = 4;
  pwdPolicy.hasDigits = YES;
  pwdPolicy.hasUpper = YES;
  pwdPolicy.hasLower = YES;
  pwdPolicy.hasSpecial = YES;
  pwdPolicy.expirationDays = 20;
  pwdPolicy.minUniqueChars = 3;
  pwdPolicy.lockTimeout = 1600;
  pwdPolicy.retryLimit = 20;

  // setPasswordPolicy will lock the vault to ensure old password
conforms to new password policy settings
  [dataVault setPasswordPolicy:pwdPolicy];

  // You must unlock the vault after setting the password policy
  [dataVault unlock:@"password!1A" Salt:@"saltD#ddg#k05%gnd[!1A"];

  // Use getPasswordPolicy to get the current policy set in the vault
  pwdPolicy = [dataVault getPasswordPolicy];
  NSLog(@" pwdPolicy %@ ",pwdPolicy.description);
```

```
  // Call setString by giving it a name:value pair to encrypt and
persist
  // a string data type within your dataVault.
  [dataVault setString:@"stringName" withValue:@"stringValue"];

  // Call getString to retrieve the string we just stored in our data
vault!
  NSString *storedStringValue = [dataVault getString:@"stringName"];
  NSLog(@" storedStringValue %@ ",storedStringValue.description);
  // Call setValue by giving it a name:value pair to encrypt and
persist
  // a binary data type within your dataVault unsigned char
acBinData[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 };
  [dataVault setValue:@"binaryName" withValue:[NSData
dataWithBytes:acBinData length:7]];

  // Call getValue to retrieve the binary we just stored in our data
vault!
  NSData *storedBinaryValue = [dataVault getValue:@"binaryName"];

  NSLog(@" storedBinaryValue %@ ",storedBinaryValue );

  // Call getDataNames to retrieve all stored element names from our
data vault
  //        NSArray * dataNames = [dataVault getDataNames];

  NSArray * dataNames = [dataVault getDataNames];

  if ( dataNames != nil ) {
    DVDataName *dataName;
    //           for ( NSInteger iIdx = 0; iIdx < [dataNames count];
iIdx++ ) {
    for ( NSInteger iIdx = 0; iIdx < [dataNames size]; iIdx ++) {
      dataName = [dataNames objectAtIndex:iIdx];
      if ( dataName.type == DVDataTypeString ) {
        // Stored value is of string type
        NSString *thisStringValue = [dataVault
getString:dataName.name];
        NSLog(@" thisStringValue %@ ",thisStringValue );
      }
      else if ( dataName.type == DVDataTypeBinary ) {
        // Stored value is of binary type
       NSData *thisBinaryValue = [dataVault getValue:dataName.name];
        NSLog(@" thisBinaryValue %@ ",thisBinaryValue );
      }
      else {
        // Unknown type. Possibly stored using previous version of
dataVault
        // Try as string first and then as binary
        NSString *thisStringValue = [dataVault
getString:dataName.name];
        if ( thisStringValue == nil ) {
          NSData *thisBinaryValue = [dataVault
getValue:dataName.name];
          NSLog(@" thisBinaryValue %@ ",thisBinaryValue );
        }
      }
    }
  }

  [dataVault changePassword:@"password!2A"
Salt:@"saltD#ddg#k05%gnd[!2A"];


  // Because this is a test example, we will delete our vault at the
end.
  // This means we will forever lose all data we persisted in our data
vault.
  [DataVault deleteVault: @"SampleVault"];
}
@catch (DataVaultException *exception)
{
  NSLog(@"Datavault exception. Reason: %@", [exception reason]);
}
```

ODP SDK API Reference for iOS

Use the ODP SDK API reference as the primary reference for all API listings and error code information.

Refer to the ODP SDK API reference for each available package.

ODPAppSettings class

Syntax

`public class  ODPAppSettings`

Remarks

Utility class to fetch settings required by the application This class implements static methods to fetch settings like end-points for application data access and for push and also connection settings like host, port and the farm

*getApplicationEndpointWithError:(NSError **) method*

Fetches the end-point using which the application can start accessing the business data (application end-point).

Syntax

`public static virtual NSString *`
`getApplicationEndpointWithError: (NSError ** ` **error**`)`

Parameters

- **error –** A double pointer to the error object in case the operation results in an error

Returns

Returns the end-point as a string object (URL/URN.) Usage NSError* error = nil; NSString* appEndPoint = [ODPAppSettings getApplicationEndpointWithError:&error]; if (!error) { // Further processing }

*getCustomizationResourceBundleWithCustomizationResource:userName:passWord:error:(NSString *, NSString *, NSString *, NSError **) method*

Returns/downloads Customization Resources as binary data.

Syntax

```
public static virtual NSData *
getCustomizationResourceBundleWithCustomizationResource:user
Name:passWord:error: (NSString * customizationResource, NSString *
userName, NSString * passWord, NSError ** error)
```

Parameters

- **customizationResource : –** Customization Resource
- **userName : –** user name
- **passWord : –** password
- **error –** A double pointer to the error object in case the operation results in an error. This API returns the Customization Resource zip/jar file as binary data.

Returns

This returns the Customization Resource in the form of binary data. Usage NSError* error = nil; NSData* customResourceBundleData = [ODPAppSettings getCustomizationResourceBundleWithCustomizationResource:customizationResource userName:USERNAME passWord:PASSWORD error:&error]; if (!error) { // Further processing }

*getFarmIdWithError:(NSError **) method*

Fetches the farm to which the SAP Mobile Server belongs to.

Syntax

```
public static virtual NSString * getFarmIdWithError: (NSError
** error)
```

Parameters

- **error –** A double pointer to the error object in case the operation results in an error.

Returns

Returns the farm ID as a string object Usage NSError* error = nil; NSString* farmId = [ODPAppSettings getFarmIdWithError:&error]; if (!error) { // Further processing }

*getPasswordPolicy:(NSError **) method*

Fetches the Password Policy set by the Admin in SCC for this Application.

Syntax

```
public static virtual PasswordPolicy * getPasswordPolicy:
(NSError ** error)
```

Parameters

- **error –** A double pointer to the error object in case the operation results in an error

Returns

Returns the structure containing the Password policy Usage NSError* error = nil; PasswordPolicy *dvppStruct = [ODPAppSettings getPasswordPolicy:&error]; if (!error) { // Further processing }

*getPortNumberWithError:(NSError **) method*

Fetches the server port number which has been opened up for client to connect to it.

Syntax

```
public static virtual NSInteger getPortNumberWithError:
(NSError ** error)
```

Parameters

- **error –** A double pointer to the error object in case the operation results in an error

Returns

Returns the server port number as an integer Usage NSError* error = nil; NSInteger portNum = [ODPAppSettings getPortNumberWithError:&error]; if (!error) { // Further processing }

*getPushEndpointWithError:(NSError **) method*

Fetches the end-point using which the enterprise back-end can push data to the ODP client via the server.

Syntax

```
public static virtual NSString * getPushEndpointWithError:
(NSError ** error)
```

Parameters

- **error –** A double pointer to the error object in case the operation results in an error

Returns

Returns the end-point as a string object (URL/URN.) Usage NSError* error = nil; NSString* pushEndPoint = [ODPAppSettings getPushEndpointWithError:&error]; if (!error) { // Further processing }

*getServerNameWithError:(NSError **) method*

Fetches the server host name against which the user is registered.

Syntax

```
public static virtual NSString * getServerNameWithError:
(NSError ** error)
```

Parameters

- **error –** A double pointer to the error object in case the operation results in an error

Returns

Returns the server host name as a string object Usage NSError* error = nil; NSString* hostName = [ODPAppSettings getServerNameWithError:&error]; if (!error) { // Further processing }

isServerKeyProvisioned() method

Checks to see the public key of the SAP Mobile Server is provisioned on the client or not.

Syntax

```
public static virtual BOOL isServerKeyProvisioned ()
```

Returns

Returns a BOOL indicating whether the key has been provisioned or not. Usage if ([ODPAppSettings isServerKeyProvisioned) { // Processing....... }

ODPCertificateManager class

Syntax

```
public class ODPCertificateManager
```

Remarks

Consists of methods to retrieve the provisioned certifice locally or remotely. This class has a static method to get the certificate as a base-64 encoded string from a local file system.

*getSignedCertificateFromFile:password:error:(NSString *, NSString *, NSError **) method*

Retrieves the X.509 certificate stored in the local file system as a base-64 encoded string.

Syntax

`public static virtual NSString * getSignedCertificateFromFile:password:error: (NSString *` **filePath**`, NSString *` **certPassword**`, NSError **` **error**`)`

Parameters

- **filePath –** Path to the file including the file name and extension.
- **certPassword –** Password for the certificate.
- **error –** Pointer to the error object to be filled by the method during an error.

Returns

x.509 certificate as a base-64 encoded string. Usage NSError* error = nil; [ODPCertificateManager getSignedCertificateFromFile:"Filename" password:@"Password" error:&error]; if (!error) { // Further Processing }

ODPClientConnection class

Syntax

```
public class ODPClientConnection
```

Remarks

Consists of methods used to interact with the underlying client to register connection settings, online payload push and APNS push notifications. Application can register for push (payload) from the back-end gateway, when the client is online, or for APNS when the application is closed or client is offline.

clearServerVerificationKey() method

For a device to switch connection between SAP Mobile Platform servers, this API is invoked before registering a user with new server.

Syntax

```
public static virtual void clearServerVerificationKey ()
```

Usage

```
This ensures that the server public keys are removed from the SAP
Mobile Platform client SDK which enables connectivity to new SAP
Mobile Platform server.
 Usage
```

```
[ODPClientConnection clearServerVerificationKey];
```

*deviceTokenForPush:deviceToken:(UIApplication *, NSData *) method*

Registers for push notification using device token as a parameter.

Syntax

```
public static virtual void deviceTokenForPush:deviceToken:
(UIApplication * application, NSData * devToken)
```

Parameters

- **application –** Application object received as a part of the delegate.
- **devToken –** Device token received from the APNS gateway after the application has registered for APNS. Usage
  - (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken { [ODPClientConnection deviceTokenForPush:application deviceToken:deviceToken]; }

*getInstance:(NSString *) method*

Get an instance of the ODPClientConnection class.

Syntax

```
public static virtual ODPClientConnection * getInstance:
(NSString * applicationId)
```

Parameters

- **applicationId –** Application ID of the application.

Returns

Singleton instance of the ODPClientConnection class. Usage ODPClientConnection* clientConnection = [ODPClientConnection getInstance:"ApplicationId"];

getPushDelegate() method

Syntax

```
public static virtual id< ODPPushDelegate > getPushDelegate
()
```

*pushNotification:notifyData:(UIApplication *, NSDictionary *) method*
Notify the application of the incoming push notification.

Syntax

`public static virtual void pushNotification:notifyData: (UIApplication *` **application**`, NSDictionary *` **userInfo**`)`

Parameters

- **application –** Application object received as a part of the delegate.
- **userInfo –** User information dictionary with additional data about the push notification. Usage
    - (void)application:(UIApplication *)application didReceiveRemoteNotification: (NSDictionary *)userInfo { [ODPClientConnection pushNotification:application notifyData:userInfo]; }

*pushRegistrationFailed:errorInfo:(UIApplication *, NSError *) method*
Reports an error if push registration to APNS fails.

Syntax

`public static virtual void pushRegistrationFailed:errorInfo: (UIApplication *` **application**`, NSError *` **err**`)`

Parameters

- **application –** Application object received as a part of the delegate.
- **err –** Error object received as a part of the delegate. Usage
    - (void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError:(NSError *)error { [ODPClientConnection pushRegistrationFailed:application errorInfo:error]; }

registerForPayloadPush:(id< ODPPushDelegate >) method
Provides information about the class that has implemented the push delegate listener.

Syntax

`public static virtual void registerForPayloadPush: (id< ODPPushDelegate >` **delegate**`)`

Parameters

- **delegate –** Instance of the class that has implemented the push listener delegate. Usage [ODPClientConnection registerForPayloadPush:self];

*setConnectionPropertiesWithHost:port:farm:user:code:(NSString *, NSInteger, NSString *, NSString *, NSString *) method*

Syntax

`public virtual void setConnectionPropertiesWithHost:port:farm:user:code: (NSString *` **hostName**`, NSInteger` **portNum**`, NSString *` **farmId**`, NSString *` **userName**`, NSString *` **activationCode**`)`

*setupForPush:(UIApplication *) method*

Registers the applicaton for APNS push notification.

Syntax

`public static virtual void setupForPush: (UIApplication *` **application**`)`

Parameters

- **application –** Application object received as a part of the delegate has to be passed to the function call. Usage
  - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions { [ODPClientConnection setupForPush:application]; }

startClient() method

Syntax

`public virtual void startClient ()`

stopClient() method

Stops all the client network traffic and the state of the client is set to sleep.

Syntax

`public virtual void stopClient ()`

Usage

```
Usage
```

```
 ODPClientConnection* clientConnection = [ODPClientConnection
getInstance:"ApplicationId"];
 [clientConnection stopClient];
```

ODPClientConnectionException class

Syntax
`public class ODPClientConnectionException`

*initWithName:reason:code:text:(NSString *, NSString *, NSString *, NSString *) method*

Syntax
`public virtual id initWithName:reason:code:text: (NSString *` **aName**`, NSString *` **aReason**`, NSString *` **errorCode**`, NSString *` **errorText**`)`

ODPClientListeners class

Syntax
`public class ODPClientListeners`

Remarks
Consists of methods used to register listeners for notifying client side changes. Register notification listeners for client state change notifications. The client state changes include connection state changes and configuration property state changes.

certificateChallengeResult:(BOOL) method

Syntax
`public static virtual void certificateChallengeResult: (BOOL` **challengeResult**`)`

Parameters

- **challengeResult –** BOOL indicating whether the certificate should be accepted. Usage [ODPClientListeners certificateChallengeResult:YES];

getCertificateChallengeListenerDelegate() method

Syntax
```
public static virtual id<
ODPCertificateChallengeListenerDelegate >
getCertificateChallengeListenerDelegate ()
```

Returns
Instance of the listener object.

getConfigurationStateListenerDelegate() method

Syntax
```
public static virtual id< ODPClientConfigurationStateListener
> getConfigurationStateListenerDelegate ()
```

Returns
Instance of the object implementing the listener interface. Usage
id<ODPClientConfigurationStateListener> = [ODPClientListeners
ODPClientConfigurationStateListener];

getConnectionStateListenerDelegate() method

Syntax
```
public static virtual id< ODPClientConnectionStateListener >
getConnectionStateListenerDelegate ()
```

Returns
Instance of the object implementing the listener interface. Usage
id<ODPClientConnectionStateListener> = [ODPClientListeners
getConnectionStateListenerDelegate];

getHTTPAuthChallengeListenerDelegate() method

Syntax
```
public static virtual id<
ODPHTTPAuthChallengeListenerDelegate >
getHTTPAuthChallengeListenerDelegate ()
```

Returns
Instance of the listener object.

getHTTPErrorListenerDelegate() method

Syntax
```
public static virtual id< ODPHTTPErrorListenerDelegate >
getHTTPErrorListenerDelegate ()
```

Returns
Instance of the listener object.

*httpAuthChallengeResult:forUser:withPassword:(BOOL, NSString *, NSString *) method*

Syntax

`public static virtual void httpAuthChallengeResult:forUser:withPassword: (BOOL` **credentialsSupplied**`, NSString *` **userName**`, NSString *` **password**`)`

Parameters

- **credentialsSupplied –** BOOL indicating whether the credentials are supplied.
- **userName –** NSString containing the username.
- **password –** NSString containing the password. Usage [ODPClientListeners httpAuthChallengeResult:YES forUser:"mobile" withPassword:@"Ntwatch"];

setCertificateChallengeListenerDelegate:(id< ODPCertificateChallengeListenerDelegate >) method

Syntax

`public static virtual void setCertificateChallengeListenerDelegate: (id< ODPCertificateChallengeListenerDelegate >` **listener**`)`

Parameters

- **listener –** A listener object whose definition adheres to the appropriate protocol. Usage [ODPClientListeners setCertificateChallengeListenerDelegate:self];

setConfigurationStateListenerDelegate:(id< ODPClientConfigurationStateListener >) method

Syntax

`public static virtual void setConfigurationStateListenerDelegate: (id< ODPClientConfigurationStateListener >` **_listener**`)`

Parameters

- **listener –** A listener object whose definition adheres to the appropriate protocol. Usage [ODPClientListeners setConfigurationStateListenerDelegate:self]; // The class implementing this should ahere to <ODPClientConfigurationStateListener> Protocol

setConnectionStateListenerDelegate:(id< ODPClientConnectionStateListener >) method

Syntax
```
public static virtual void
setConnectionStateListenerDelegate: (id<
ODPClientConnectionStateListener > _listener)
```

Parameters

- **listener –** A listener object whose definition adheres to the appropriate protocol. Usage [ODPClientListeners setConnectionStateListenerDelegate:self]; // The class implementing this should ahere to <ODPClientConnectionStateListener> Protocol

setHTTPAuthChallengeListenerDelegate:(id< ODPHTTPAuthChallengeListenerDelegate >) method

Syntax
```
public static virtual void
setHTTPAuthChallengeListenerDelegate: (id<
ODPHTTPAuthChallengeListenerDelegate > listener)
```

Parameters

- **listener –** A listener object whose definition adheres to the appropriate protocol. Usage [ODPClientListeners setHTTPAuthChallengeListenerDelegate:self];

setHTTPErrorListenerDelegate:(id< ODPHTTPErrorListenerDelegate >) method

Syntax
```
public static virtual void  setHTTPErrorListenerDelegate: (id<
ODPHTTPErrorListenerDelegate > listener)
```

Parameters

- **listener –** A listener object whose definition adheres to the appropriate protocol. Usage [ODPClientListeners setHTTPErrorListenerDelegate:self];

ODPRequest class
Consists of methods for handling HTTP requests.

Syntax
```
public class  ODPRequest
```

Remarks
Wrapper around CFNetwork Features:

- Synchronous and asynchronous execution
- Continues downloading and uploading when app is sent to the background (iOS 4.0+ only)
- Network Queue for batch http request processing and progress tracking
- Protected download cache; encrypted offline storage of downloaded content (iOS 4.0+ only)
- Secure connection (iOS 4.0+ only)
- Easy to extend
- Progress tracking

*addBasicAuthenticationHeaderWithUsername:andPassword:(NSString *, NSString *) method*
Constructs a basic authentication header from the user name and password specified, and adds it to the request headers.

Syntax
`public virtual void addBasicAuthenticationHeaderWithUsername:andPassword: (NSString *` **theUsername**`, NSString *` **thePassword**`)`

Usage

```
It is used when the variable shouldPresentCredentialsBeforeChallenge
is set to YES.
Usage

                     ODPRequest *request=[ODPRequest requestWithURL:
[NSURL URLWithString:endpointUrl]];
[request addBasicAuthenticationHeaderWithUsername:"smpuser"
andPassword:@"s3puser"];
[request startSynchronous];
```

*addRequestHeader:value:(NSString *, NSString *) method*
Adds a custom header to the request.

Syntax
`public virtual void addRequestHeader:value: (NSString *` **header**`, NSString *` **value**`)`

Parameters

- **header –** Header name.

- **value –** Header value. Usage

*appendPostData:(NSData *) method*

Adds data to the post body.

Syntax

`public virtual void appendPostData: (NSData * `**`data`**`)`

Usage

```
Appends to postBody when shouldStreamPostDataFromDisk is set to
false, or write to postBodyWriteStream when set to true.
Usage

                     ODPRequest *request=[ODPRequest requestWithURL:
[NSURL URLWithString:endpointUrl]];
[request setUsername:"smpuser"];
[request setPassword:@"s3puser"];
[request setRequestMethod:@"POST"];
[request appendPostData:data];
[request startSynchronous];
```

*appendPostDataFromFile:(NSString *) method*

Adds data to the post body from a file.

Syntax

`public virtual void appendPostDataFromFile: (NSString * `**`file`**`)`

Usage

```
Appends to postBody when shouldStreamPostDataFromDisk is set to
false, or writes to postBodyWriteStream when set to true.
Usage

NSString *documentsDirectory = [NSHomeDirectory()
stringByAppendingPathComponent:"Documents"];
NSString *filePath = [documentsDirectory
stringByAppendingPathComponent:@"uploadData.xml"];
ODPRequest *request=[ODPRequest requestWithURL:[NSURL
URLWithString:endpointUrl]];
[request setUsername:"smpuser"];
[request setPassword:@"s3puser"];
[request setRequestMethod:@"POST"];
[request appendPostDataFromFile:filePath];
[request startSynchronous];
```

applyAuthorizationHeader() method

Syntax

`public virtual void applyAuthorizationHeader ()`

applyCookieHeader() method
Creates a cookie header from request cookies and global store, when called during buildRequestHeaders after a redirect.

Syntax
```
public virtual void applyCookieHeader ()
```

averageBandwidthUsedPerSecond() method
Returns an average (for the last five seconds) of how much bandwidth is used, in bytes.

Syntax
```
public static virtual unsigned long
averageBandwidthUsedPerSecond ()
```

*base64forData:(NSData *) method*
Base64 encodes the provided NSData*.

Syntax
```
public static virtual NSString * base64forData: (NSData *
theData)
```

buildPostBody() method
Creates the post body.

Syntax
```
public virtual void buildPostBody ()
```

buildRequestHeaders() method
Populates the request headers dictionary.

Syntax
```
public virtual void buildRequestHeaders ()
```

Usage

It is called before a request is started, or by a HEAD request to borrow the headers.

*bumpToTextFile:(NSString *) method*

Syntax
```
public static virtual void bumpToTextFile: (NSString * xml)
```

cancelAuthentication() method
Called by delegates to cancel authentication and stop.

Syntax
```
public virtual void cancelAuthentication ()
```

clearSession() method

Syntax
```
public static virtual void clearSession ()
```

dataWritingOptions() method
Returns a mask which determines whether the data should be stored in encrypted form.

Syntax
```
public static virtual const NSDataWritingOptions
dataWritingOptions ()
```

Usage

Data is stored in encrypted form by default.

*dateFromRFC1123String:(NSString *) method*
Returns a date from a string in RFC1123 format.

Syntax
```
public static virtual NSDate * dateFromRFC1123String:
(NSString * string)
```

*decodeBase64:(NSString *) method*
Decodes a base64 encoded string.

Syntax
```
public static virtual NSString * decodeBase64: (NSString *
string_in)
```

defaultCache() method
Returns the default download cache or nil if no value is set at class level.

Syntax
```
public static virtual id< SDMCacheDelegate > defaultCache ()
```

enableXCSRF:(BOOL) method
Enables XCSRF support.

Syntax
```
public static virtual void enableXCSRF: (BOOL enable)
```

Parameters

- **Boolean –** for enabling or disabling the XCSRF support. Usage

*failWithError:(NSError *) method*
Called when a request fails, and lets the delegate know via didFailSelector.

Syntax
```
public virtual void failWithError: (NSError * theError)
```

fileProtectionAttributes() method
Returns an attributes dictionary used for encrypted file storage or nil if file protection has been explicitly disabled.

Syntax
```
public static virtual NSDictionary * fileProtectionAttributes
()
```

Usage

File protection is enabled by default for builds targeting iOS4.0 and above. You can change it using SDMHttpRequest +setProtectionEnabled.

getChunkSize() method

Syntax
```
public virtual int getChunkSize ()
```

getMaxConcurrentHTTPRequestCount() method
Returns the maximum number of ODPRequest threads which can be executed in parallel.

Syntax
```
public static virtual NSInteger
getMaxConcurrentHTTPRequestCount ()
```

getMoOffset() method

Syntax

```
public virtual unsigned long getMoOffset ()
```

handleNetworkEvent:(CFStreamEventType) method

Handles network events as they occur.

Syntax

```
public virtual void handleNetworkEvent: (CFStreamEventType
type)
```

Usage

It is called from ReadStreamClientCallBack method when one of the events requested in NETWORK_EVENTS occurs.

HEADRequest() method

Used by NetworkQueue to create a HEAD request appropriate for this request with the same headers.

Syntax

```
public virtual ODPRequest * HEADRequest ()
```

hideNetworkActivityIndicator() method

Hides the network activity spinner.

Syntax

```
public static virtual void hideNetworkActivityIndicator ()
```

incrementBandwidthUsedInLastSecond:(unsigned long) method

Records bandwidth use and used by InputStreams during upload.

Syntax

```
public static virtual void
incrementBandwidthUsedInLastSecond: (unsigned long bytes)
```

incrementDownloadSizeBy:(long long) method
Called when you get a content-length header and shouldResetDownloadProgress is set to true.

Syntax
`public virtual void incrementDownloadSizeBy: (long long` **length**`)`

incrementUploadSizeBy:(long long) method
Called when a request starts and shouldResetUploadProgress is set to true.

Syntax
`public virtual void incrementUploadSizeBy: (long long` **length**`)`

Usage

It is also called (with a negative length) to remove the size of the underlying buffer used for upload.

*initWithURL:(NSURL *) method*
HTTP or HTTPS URL.

Syntax
`public virtual id initWithURL: (NSURL *` **newURL**`)`

Usage

It may include username and password if appropriate.

isBandwidthThrottled() method
Returns YES if bandwidth throttling is currently in use.

Syntax
`public static virtual BOOL isBandwidthThrottled ()`

isNetworkInUse() method
Indicates whether there are running requests.

Syntax
`public static virtual BOOL isNetworkInUse ()`

Returns
NO, if no requests are currently executing.

isNetworkReachableViaWWAN() method
Returns YES when an iPhone OS device is connected via WWAN, false when connected via WIFI or not connected.

Syntax
```
public static virtual BOOL isNetworkReachableViaWWAN ()
```

isResponseCompressed() method
Returns true if the response is gzip compressed.

Syntax
```
public virtual BOOL isResponseCompressed ()
```

Returns
True or false depending on whether the response is gzip or not. Usage

maxBandwidthPerSecond() method
Maximum approximate number of bytes all requests can send or receive in a second.

Syntax
```
public static virtual unsigned long maxBandwidthPerSecond ()
```

Usage

This does not include HTTP headers.

maxUploadReadLength() method
Returns the maximum amount of data that can be read as part of the current measurement period, and retires the thread if the allowance is consumed.

Syntax
```
public static virtual unsigned long maxUploadReadLength ()
```

*mimeTypeForFileAtPath:(NSString *) method*

Syntax
```
public static virtual NSString * mimeTypeForFileAtPath:
(NSString * path)
```

*performSelector:onTarget:withObject:amount:(SEL, id *, id, void *) method*
Helper method used for performing invocations on the main thread (used for progress).

Syntax
public static virtual void performSelector:onTarget:withObject:amount: (SEL **selector**, id * **target**, id **object**, void * **amount**)

performThrottling() method
Performs bandwidth throttling.

Syntax
public virtual void performThrottling ()

*removeCredentialsForHost:port:protocol:realm:(NSString *, int, NSString *, NSString *) method*
Deletes host credentials from the keychain.

Syntax
public static virtual void removeCredentialsForHost:port:protocol:realm: (NSString * **host**, int **port**, NSString * **protocol**, NSString * **realm**)

*removeCredentialsForProxy:port:realm:(NSString *, int, NSString *) method*
Deletes proxy credentials from the keychain.

Syntax
public static virtual void removeCredentialsForProxy:port:realm: (NSString * **host**, int **port**, NSString * **realm**)

removeUploadProgressSoFar() method
Called when authorization is required, as you can determine that you do not have permissions only when the upload is complete.

Syntax
public virtual void removeUploadProgressSoFar ()

*requestWithURL:(NSURL *) method*
Convenience constructor with URL.

Syntax

`public static virtual id requestWithURL: (NSURL *` **newURL**`)`

Parameters

- **URL –**

Returns

ODPRequest object Usage ODPRequest * l_request=[ODPRequest requestWithURL: [NSURL URLWithString:Url]]; .... add the necessary request headers

*requestWithURL:usingCache:(NSURL *, id< SDMCacheDelegate >) method*
Convenience constructor with URL and cache.

Syntax

`public static virtual id requestWithURL:usingCache: (NSURL *` **newURL**`, id< SDMCacheDelegate >` **cache**`)`

Parameters

- **URL –**
- **Cache –**

Returns

ODPRequest object Usage ODPRequest * l_request=[ODPRequest requestWithURL: [NSURL URLWithString:Url] usingCache:cache]; .... add the necessary request headers

*requestWithURL:usingCache:andCachePolicy:(NSURL *, id< SDMCacheDelegate >, CachePolicy) method*
Convenience constructor with URL, cache, and cache policy.

Syntax

`public static virtual id requestWithURL:usingCache:andCachePolicy: (NSURL *` **newURL**`, id< SDMCacheDelegate >` **cache**`, CachePolicy` **policy**`)`

Parameters

- **URL –**

- **Cache –**

Returns
ODPRequest object. Usage ODPRequest * l_request=[ODPRequest requestWithURL: [NSURL URLWithString:Url] usingCache:cache andCachePolicy:policy]; .... add the necessary request headers

responseData() method
Response data, automatically uncompressed where appropriate.

Syntax
```
public virtual NSData *  responseData ()
```

Returns
Binary data Usage

responseString() method
Returns the contents of the result as an NSString.

Syntax
```
public virtual NSString *  responseString ()
```

Returns
String data Usage

retryUsingSuppliedCredentials() method
Unlocks the request thread so it can resume the request.

Syntax
```
public virtual void  retryUsingSuppliedCredentials ()
```

Usage

This method should be called by delegates once the authentication information is populated after an authentication challenge.

*saveCredentials:forHost:port:protocol:realm:(NSURLCredential *, NSString *, int, NSString *, NSString *) method*
Saves host credentials to the keychain.

Syntax
```
public static virtual void
saveCredentials:forHost:port:protocol:realm: (NSURLCredential
```

`*` **credentials**`,` `NSString *` **host**`,` `int` **port**`,` `NSString *` **protocol**`,` `NSString *` **realm**`)`

*saveCredentials:forProxy:port:realm:(NSURLCredential *, NSString *, int, NSString *) method*
Saves proxy credentials to the keychain.

Syntax
`public static virtual void saveCredentials:forProxy:port:realm: (NSURLCredential *` **credentials**`,` `NSString *` **host**`,` `int` **port**`,` `NSString *` **realm**`)`

*saveCredentialsToKeychain:(NSDictionary *) method*
Saves credentials for this request to the keychain.

Syntax
`public virtual void saveCredentialsToKeychain: (NSDictionary *` **newCredentials**`)`

*savedCredentialsForHost:port:protocol:realm:(NSString *, int, NSString *, NSString *) method*
Returns host credentials from the keychain.

Syntax
`public static virtual NSURLCredential * savedCredentialsForHost:port:protocol:realm: (NSString *` **host**`,` `int` **port**`,` `NSString *` **protocol**`,` `NSString *` **realm**`)`

*savedCredentialsForProxy:port:protocol:realm:(NSString *, int, NSString *, NSString *) method*
Returns proxy credentials from the keychain.

Syntax
`public static virtual NSURLCredential * savedCredentialsForProxy:port:protocol:realm: (NSString *` **host**`,` `int` **port**`,` `NSString *` **protocol**`,` `NSString *` **realm**`)`

setChunkSize:(int) method

Syntax
`public virtual void setChunkSize: (int` **size**`)`

setClientCertificateIdentity:(SecIdentityRef) method
Sets the identity reference (X.509 certificate and private key) for this request. It is used for secure connections.

Syntax
```
public virtual void setClientCertificateIdentity:
(SecIdentityRef anIdentity)
```

setDefaultCache:(id< SDMCacheDelegate >) method
Configures a default download cache which applies for all requests.

Syntax
```
public static virtual void setDefaultCache: (id<
SDMCacheDelegate > cache)
```

*setEtag:withMatchType:(NSString *, EtagMatchType) method*
Adds If-Match or If-None-Match or If-Range header.

Syntax
```
public virtual void setEtag:withMatchType: (NSString * etag,
EtagMatchType matchType)
```

setMaxBandwidthPerSecond:(unsigned long) method
Sets the maximum number of bytes all requests can send or receive in a second.

Syntax
```
public static virtual void setMaxBandwidthPerSecond: (unsigned
long bytes)
```

setMaxConcurrentHTTPRequestCount:(const unsigned char) method
Sets the maximum number of ODPRequest threads that can be executed at the same time.

Syntax
```
public static virtual void setMaxConcurrentHTTPRequestCount:
(const unsigned char cnt)
```

Exceptions

- **SDMConnectivityException –** If maximum number of allowed parallel threads is exceeded.

setMoOffset:(unsigned long) method

Syntax

```
public virtual void setMoOffset: (unsigned long offset)
```

setProtectionEnabled:(BOOL) method

Controls whether the download cache should store files using protection.

Syntax

```
public static virtual void setProtectionEnabled: (BOOL flag_in)
```

Usage

It is enabled by default and applies to all instances. It is available for builds targeting iOS 4.0 or above.

setShouldThrottleBandwidthForWWAN:(BOOL) method

ODPRequest automatically turns throttling on and off as the connection type changes between WWAN and WiFi.

Syntax

```
public static virtual void setShouldThrottleBandwidthForWWAN:
(BOOL throttle)
```

Usage

Set to YES to automatically turn on throttling when WWAN is connected.

setShouldUpdateNetworkActivityIndicator:(BOOL) method

Controls whether the request should take over updating the network activity indicator.

Syntax

```
public static virtual void
setShouldUpdateNetworkActivityIndicator: (BOOL shouldUpdate)
```

Usage

It is enabled by default.

sharedQueue() method
Returns the shared queue.

Syntax
```
public static virtual NSOperationQueue * sharedQueue ()
```

showNetworkActivityIndicator() method
Shows the network activity spinner.

Syntax
```
public static virtual void showNetworkActivityIndicator ()
```

startAsynchronous() method
Executes the request in the background.

Syntax
```
public virtual void startAsynchronous ()
```

Usage
```
The client should register the callback selector to get notified
about various events such as failure, completion, and so on.
Usage

                       ODPRequest * l_request=[ODPRequest
requestWithURL:[NSURL URLWithString:endpointUrl]];
.... add the necessary request headers





                     [l_request startAsynchronous];
```

startSynchronous() method
Executes a request synchronously, and returns control when the request completes or fails.

Syntax
```
public virtual void startSynchronous ()
```

Usage
```

Usage

                       ODPRequest * l_request=[ODPRequest
requestWithURL:[NSURL URLWithString:endpointUrl]];
```

```
.... add the necessary request headers




                    [l_request startSynchronous];
```

startTrace() method
If tracing is enabled, this method starts preparing BTX.

Syntax
`public static virtual void startTrace ()`

Usage

```
Usage

{
    [ODPRequest startTrace];
 }
(NSException *e){
    // catch exception here
}
```

stopTrace() method
Disables the tracing (if enabled) and stops generating BTX.

Syntax
`public static virtual void stopTrace ()`

Usage

```
Usage

{
    [ODPRequest stopTrace];
 }
(NSException *e){
    // catch exception here
}
```

throttleBandwidthForWWANUsingLimit:(unsigned long) method
Turns on throttling automatically when WWAN is connected using a custom limit.

Syntax
`public static virtual void`
`throttleBandwidthForWWANUsingLimit: (unsigned long` **limit**`)`

updateDownloadProgress() method
Updates download progress (notifies the queue and/or downloadProgressDelegate of this request).

Syntax
```
public virtual void updateDownloadProgress ()
```

*updateProgressIndicator:withProgress:ofTotal:(id *, unsigned long long, unsigned long long) method*
Helper method for interacting with progress indicators to abstract the details of different APIS (NSProgressIndicator and UIProgressView).

Syntax
```
public static virtual void
updateProgressIndicator:withProgress:ofTotal: (id * indicator,
unsigned long long progress, unsigned long long total)
```

updateProgressIndicators() method
Updates the progress delegates.

Syntax
```
public virtual void updateProgressIndicators ()
```

Usage

Used by SDMNetworkQueue, and should not be invoked directly by clients.

updateUploadProgress() method
Updates upload progress (notifies the queue and/or uploadProgressDelegate of this request).

Syntax
```
public virtual void updateUploadProgress ()
```

*uploadTraceWithError:(NSError **) method*

Syntax
```
public static virtual void uploadTraceWithError: (NSError **
error)
```

Parameters

- **error : –** Double pointer to the error object if the function returns an error. Uploads the generated BTX to Solution Manager. If the method fails, it will throw an error.

ODPUserManager class

Syntax

```
public class  ODPUserManager
```

Remarks

Consists of methods for registering and de-registering users. All the function calls in this class have various ways of provisioning settings on to the client and register the user with the help of these settings. We can also check if a user is registered and delete the user as and when necessary

Error codes for registration

515 - Error retrieving the public key from the server. 516 - Error sending request to the server. 514 - Unable to start traveler session. 516 - Resetting communications or communications cancelled during send. 558 - Cannot connect to server. 563 - Bad credentials. 571 - Server response timeout that occurs normally without ill effects. 578 - Wrong user for device. 579 - Wrong device for code. 581 - Activation code check failed. 584 - Wrong public key. 580 - Activation code is invalid. 6400 - Communication error that occurs normally without ill effects. 6600 - COMMERR_USER_CANCELLED 14801 - MCLERR_CALL_INIT_INSTANCE_FIRST 14802 - MCLERR_CONNECTION_SETTINGS_INCOMPLETE 14803 - MCLERR_INIT_INSTANCE_FAILED 14804 - MCLERR_SET_CONFIG_PROPERTY_FAILED 14805 - MCLERR_NULL_PROPERTY_VALUE 14806 - MCLERR_START_CLIENT_FAILED 14807 - MCLERR_SHUTDOWN_CLIENT_FAILED 14809 - MCLERR_MOOBECT_CREATE_FAILURE 14810 - MCLERR_OPERATION_INVALID_FOR_STATE 14811 - MCLERR_OBJECT_REGISTRATION_FAILURE 14812 - MCLERR_INVALID_PARAMETER_TYPE 14813 - MCLERR_COULD_NOT_REACH_MMS_SERVER 14814 - MCLERR_MMS_AUTHENTICATION_FAILED 14815 - MCLERR_READ_FAILURE 14816 - MCLERR_WRITE_FAILURE

Auto registration errors

14850 - MCLERR_AUTO_REG_TEMPLATE_NOT_FOUND 14851 - MCLERR_AUTO_REG_NOT_ENABLED 14852 - MCLERR_AUTO_REG_REGISTRATION_NOT_FOUND 14853 - MCLERR_AUTO_REG_WRONG_USER_FOR_DEVICE 14854 - MCLERR_AUTO_REG_USER_NAME_TOO_LONG 14855 -

MCLERR_AUTO_REG_DEVICE_ALREADY_REGISTERED 14899 - MCLERR_UNKNOWN_ERROR

70000 - One or more input fields are empty. 70001 - User is already registered or the previous user has not been deleted. 70002 - User to be deleted is not registered. 70003 - User registration timed out. 70004 - All delegate methods have not been implemented. 70005 - SAP Mobile Platform internal parsing error. 70006 - Empty response string received from the server. 70007 - Customization Bundle not available.

Error codes for request response

71000 - URL is empty. 71001 - Application identifier is null. User is not registered. 71002 - Internal error. JSON parsing has failed.

*deleteUserWithError:(NSError **) method*

Deletes the application connection or user with which the application has established a connection.

Syntax

`public virtual BOOL deleteUserWithError: (NSError ** error)`

Parameters

- **error –** Pointer to the error object if the function returns an error.

Returns

BOOL indicating if the user has been deleted or not. Usage ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; NSError* error = nil; if([userManager deleteUserWithError:&error]) { // Further Processing } else { // Further Error processing using error object }

enableHTTPS:(BOOL) method

Syntax

`public static virtual void enableHTTPS: (BOOL flag)`

Parameters

- **flag –** Indicates whether HTTPS should be used. Usage [ODPUserManager enableHTTPS:YES];

*getInstance:(NSString *) method*

Initialize the application.

Syntax

```
public static virtual ODPUserManager * getInstance: (NSString
* applicationId)
```

Parameters

- **applicationId –** Application ID of the application trying to establish the connection.

Returns

Instance of ODPUserManager class Usage ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"];

isUserRegistered() method

Checks if the user is still registered with the server and returns a value accordingly.

Syntax

```
public virtual BOOL isUserRegistered ()
```

Returns

BOOL indicating if the user is registered or not. Usage ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; if([userManager isUserRegistered]) { // Further Processing }

*registerUser:activationCode:error:isSyncFlag:(NSString *, NSString *, NSError **, BOOL) method*

Registers the client with a previous manually white-listed user on SCC.

Syntax

```
public virtual BOOL
registerUser:activationCode:error:isSyncFlag: (NSString *
userName, NSString * activationCode, NSError ** error, BOOL
isSynchronous)
```

Parameters

- **userName –** User Name that is white-listed on SCC.
- **activationCode –** Activation code of the user created in SCC.
- **error –** Pointer to the error object if the function returns an error.

- **isSynchronous –** Flag to indicate if the user registration is synchronous or ansynchronous.

Returns

BOOL indicating if the user registration has been successful. Significant in case of synchronous registration only. Usage (Synchronous registration) ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; [userManager setConnectionProfileWithHost:@"hostname" port:5001 farm:@"farmid" error:nil]; NSError* error = nil; if ([userManager registerUser:"username" activationCode:@"actcode" error:&error isSyncFlag:YES]) { // Further processing } else { // Error handling using the error object defined above } Usage (Asynchronous Registration) ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; [userManager setConnectionProfileWithHost:@"hostname" port:5001 farm:@"farmid" error:nil]; [userManager setDelegate:self]; // The class implementing the listener methods should adhere to <ODPUserManagerDelegate> protocol [userManager registerUser:"username" activationCode:@"actcode" error:nil isSyncFlag:NO];

*registerUser:securityConfig:password:error:isSyncFlag:(NSString *, NSString *, NSString *, NSError **, BOOL) method*

Registers the client with the server by automatically creating a user with the help of a pre-defined authentication mechanism.

Syntax

`public virtual BOOL registerUser:securityConfig:password:error:isSyncFlag:(NSString *` **userName**`, NSString *` **securityConfig**`, NSString *` **loginPassword**`, NSError **` **error**`, BOOL` **isSynchronous**`)`

Parameters

- **userName –** User Name authorized by a predefined security provider.
- **securityConfig –** Security configuration name authenticating the user against an authentication provider.
- **loginPassword –** Password for the user (certificate string in case of certificate registration).
- **error –** Pointer to the error object if the function returns an error.
- **isSynchronous –** Flag to indicate if the user registration is synchronous or ansynchronous.

Returns

BOOL indicating if the user registration has been successful. Significant in case of synchronous registration only. Usage (Synchronous registration) ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; [userManager setConnectionProfileWithHost:@"hostname" port:5001 farm:@"farmid" error:nil];

NSError* error = nil; if ([userManager registerUser:"username" securityConfig:@"securityconfig" password:@"password" error:&error isSyncFlag:YES]) { // Further processing } else { // Error handling using the error object defined above } Usage (Asynchronous Registration) ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; [userManager setConnectionProfileWithHost:@"hostname" port:5001 farm:@"farmid" error:nil]; [userManager setDelegate:self]; // The class implementing the listener methods should adhere to <ODPUserManagerDelegate> protocol [userManager registerUser:"username" securityConfig:@"securityconfig" password:@"password" error:nil isSyncFlag:NO];

*setConnectionProfileWithHost:port:farm:error:(NSString *, NSInteger, NSString *, NSError **) method*

Provides connection properties to the UserManager class before registration or changes the connection settings after the user is registered.

Syntax

```
public virtual BOOL
setConnectionProfileWithHost:port:farm:error: (NSString *
hostName, NSInteger portNumber, NSString * farmId, NSError **
error)
```

Parameters

- **hostName –** Host name or IP Address of the SAP Mobile Server.
- **portNumber –** Port number of the SAP Mobile Server with which the client communicates.
- **farmId –** Relay server farm ID to which the SAP Mobile Server belongs. Relevant only during registration through relay server.
- **error –** Double pointer to the error object if the function returns an error. This is relevant while making changes to connection settings after the user has been registered.

Returns

BOOL indicating if the operation is successful or not. Usage (Connection settings before user registration) ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; [userManager setConnectionProfileWithHost:@"hostname" port:5001 farm:@"farmid" error:nil]; Usage (Connection settings change after user registration) ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; NSError* error = nil; [userManager setConnectionProfileWithHost:"hostname" port:5001 farm:@"farmid" error:&error];

*setHttpHeaders:cookies:error:(NSDictionary *, NSDictionary *, NSError **) method*
Set the HTTP headers and cookies to the HTTP request made from the client to the server.

Syntax

`public virtual void setHttpHeaders:cookies:error: (NSDictionary * `**headers**`, NSDictionary * `**cookies**`, NSError ** `**error**`)`

Parameters

- **headers –** HTTP header list as a key-value pair.
- **cookies –** HTTP cookie list as a key-value pair. Usage ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; `* NSDictionary* headers = [NSDictionary dictionaryWithObject:@"sample value" forKey:@"Sample-Header"]; NSDictionary* cookies = [NSDictionary dictionaryWithObject:@"adfasdJZBFSFJzjbZZBoZFPAKDJPHJZXN" forKey:@"MYSAPSSO2"]; [userManager setHttpHeaders:headers error:cookies]; OR [userManager setHttpHeaders:nil error:cookies]; // If there are only cookies to be passed OR [userManager setHttpHeaders:headers error:nil]; // If there are only headers to be passed // Further Processing

*setRelayServerUrlTemplate:error:(NSString *, NSError **) method*
Sets the URL suffix configured for the relay server.

Syntax

`public virtual void setRelayServerUrlTemplate:error: (NSString * `**_urlSuffix**`, NSError ** `**error**`)`

Parameters

- **_urlSuffix –** URL suffix to be configured for the relay server.
- **error –** Double pointer to the error object. Usage ODPUserManager* userManager = [ODPUserManager getInstance:"ApplicationId"]; NSError *error =nil,*error1=nil; [userManager setConnectionProfileWithHost:"hostName" port:[portNumber intValue] farm:@"farmId" error:&error]; [userManager setRelayServerUrlTemplate:@"urlSuffix" error:&error1]; // Further Processing

ODPCertificateChallengeListenerDelegate protocol

Syntax

```
public protocol ODPCertificateChallengeListenerDelegate
```

Remarks
Protocol for classes implementing listeners for certificate challenges. The SAP Mobile Platform library should be able to connect to a relay server via HTTPS (or HTTP) using the iPhone SDKs native connectivity APIs. If HTTPS is used, and a connection attempt occurs, some negotiation takes place. Depending on what SSL certificate is installed on the server, a different sequence can occur. A certificate is delivered from the server to the device. If the certificate is trusted by the system or application keychain, then the connection is established. Otherwise, a MessagingClientLibrary delegate is called, if there is a listener registered to handle it, to determine whether the certificate should be trusted.

*onCertificateChallenge:(NSString *) method*
Called when the ODPCertificateChallengeListenerDelegate protocol is implemented by the class.

Syntax
`public virtual void onCertificateChallenge: (NSString *` **certInfo**`)`

Usage
```
The application should implement this method to receive the
certificate information.
 Usage

-(void) onCertificateChallenge:certInfo
{
     NSLog("Cert Info is: %", certInfo);
     [ODPClientListeners certificateChallengeResult:YES];
}
```

ODPClientConfigurationStateListener protocol

Syntax
`public protocol ODPClientConfigurationStateListener`

Remarks
Protocol for classes implementing listeners for client configuration state changes. This can correspond to changing properties on the SCC such as host, port, endpoint details etc. Follow the protocol and implement the required method to listen to property value changes initiated from the server or client. Property Ids 1 ~~
MCL_PROP_ID_CONNECTION_SERVER_NAME 2 ~~
MCL_PROP_ID_CONNECTION_SERVER_PORT 3 ~~
MCL_PROP_ID_CONNECTION_FARM_ID 7 ~~
MCL_PROP_ID_CONNECTION_DOMAIN 8 ~~
MCL_PROP_ID_CONNECTION_SYNC_SVR_HOST 9 ~~
MCL_PROP_ID_CONNECTION_SYNC_SVR_PORT 10 ~~

MCL_PROP_ID_CONNECTION_SYNC_SVR_PROTOCOL 11 ~~ MCL_PROP_ID_CONNECTION_SYNC_SVR_URL_SUFFIX 12 ~~ MCL_PROP_ID_CONNECTION_SYNC_SVR_STREAM_PARAMS 20 ~~ MCL_PROP_ID_CONNECTION_USE_HTTPS 10001 ~~ MCL_PROP_ID_CONNECTION_USER_NAME 10002 ~~ MCL_PROP_ID_CONNECTION_ACTIVATION_CODE 1200 ~~ MCL_PROP_ID_DEVICE_MODEL 1201 ~~ MCL_PROP_ID_DEVICE_SUBTYPE 1202 ~~ MCL_PROP_ID_DEVICE_PHONE_NUMBER 1203 ~~ MCL_PROP_ID_DEVICE_IMSI 1302 ~~ MCL_PROP_ID_ADVANCED_MOCA_TRACE_LEVEL 1303 ~~ MCL_PROP_ID_ADVANCED_MOCA_TRACE_SIZE 1305 ~~ MCL_PROP_ID_ADVANCED_RELAY_SVR_URL_TEMPLATE 2300 ~~ MCL_PROP_DEF_CUSTOM_CUSTOM1 2301 ~~ MCL_PROP_DEF_CUSTOM_CUSTOM2 2302 ~~ MCL_PROP_DEF_CUSTOM_CUSTOM3 2303 ~~ MCL_PROP_DEF_CUSTOM_CUSTOM4 2400 ~~ MCL_PROP_ID_IPHONE_SEC_MIN_PASSWORD_LENGTH 2401 ~~ MCL_PROP_ID_IPHONE_SEC_REQUIRE_STRONG_PASSWORD 2402 ~~ MCL_PROP_ID_IPHONE_SEC_IDLE_TIMEOUT 2403 ~~ MCL_PROP_ID_IPHONE_SEC_MISSED_PASSWORD_DATA_WIPE 2800 ~~ MCL_PROP_ID_SECURITY_E2E_ENCRYPTION_ENABLED 2801 ~~ MCL_PROP_ID_SECURITY_E2E_ENCRYPTION_TYPE 2802 ~~ MCL_PROP_ID_SECURITY_TLS_TYPE 1400 ~~ MCL_PROP_ID_FT_SM_LEVEL 3000 ~~ MCL_PROP_ID_PROXY_APPLICATION_ENDPOINT 3001 ~~ MCL_PROP_ID_PROXY_PUSH_ENDPOINT 10005 ~~ MCL_PROP_ID_CONNECTION_AUTO_REGISTRATION_HINT 20000 ~~ MCL_PROP_ID_SETTINGS_EXCHANGE_COMPLETE

onConfigurationChange:value:(NSInteger, id) method
Called when there are client configuration property state changes, from the underlying layer.

Syntax

`public virtual void onConfigurationChange:value: (NSInteger` **iPropertyID**`, id` **oValue**`)`

Usage

```
 Usage

 -(void)onConfigurationChange:(NSInteger)iPropertyID  value:
(id)oValue
{
     NSLog("--d---%@---", iPropertyID, oValue);
     if(iPropertyID==kCustResourceBundlePropID)
{
     //Handle customization resource bundle
}
}
```

ODPClientConnectionStateListener protocol

Syntax

`public protocol  ODPClientConnectionStateListener`

Remarks

Protocol for classes implementing listeners for client connection state changes. Adhere to the protocol and implement the required method to listen to client connection state changes and error codes and notifies if there is a problem with the connection.

Connection Status Codes 1 -- Connected 2 -- Disconnected 3 -- Flight Mode 4 -- Out of network coverage 5 -- Waiting to connect 6 -- Device Roaming 7 -- Device Low Storage

*onConnectionStateChanged:connectionType:error:errorMsg:(NSInteger, NSInteger, NSInteger, NSString *) method*

Called when there is a client connection state change from the underlying layer.

Syntax

public virtual void
onConnectionStateChanged:connectionType:error:errorMsg:
(NSInteger **connStatus**, NSInteger **connType**, NSInteger **errCode**,
NSString * **errMsg**)

Usage

```
Implement the method to handle all connection state change
notifications.
 Usage

 -(void) onConnectionStateChanged:(NSInteger) connStatus
connectionType:(NSInteger)connType error:(NSInteger)errCode
errorMsg:(NSString *)errMsg
{
    NSLog("d--d--d--%", connStatus, connType, errCode, errMsg);
}
```

ODPHTTPAuthChallengeListenerDelegate protocol

Syntax

`public protocol  ODPHTTPAuthChallengeListenerDelegate`

Remarks

Protocol for classes implementing listeners for HTTP authentication challenges. Optionally, the host that the client is connecting to (like a relay server) requires the client to authenticate with it. The method of authentication to be supported is basic authentication. On receiving an unauthorized request from a client, the server will respond with a challenge HTTP header: “WWW-Authenticate: Basic realm=”SAP Relay Server””. The client should send a user ID and password, separated by a single colon character within a base 64 encoded string for credentials within a subsequent HTTP request. It will send the same credentials with all subsequent requests until another challenge is received. While this is supported across HTTP and HTTPS, basic authentication across HTTP is considered to be non-secure, as the base 64 encoded string is essentially the same thing as plain text. If a user switches from HTTPS to HTTP, the basic authentication credentials is cleared.

*onHTTPAuthChallenge:forUser:withRealm:(NSString *, NSString *, NSString *) method*

Called when ODPHTTPAuthChallengeListenerDelegate protocol is implemented by the class.

Syntax

public virtual void onHTTPAuthChallenge:forUser:withRealm:(NSString * **host**, NSString * **userName**, NSString * **realm**)

Usage

```
User Name contains the previous value on input.
 Usage

 -(void) onHTTPAuthChallenge:(NSString*)host forUser:
(NSString*)userName withRealm:(NSString*)realm
{
     NSLog("%@--%@--%", host, userName, realm);
     [ODPClientListeners httpAuthChallengeResult:YES
forUser:"mobile" withPassword:@"Ntwatch"];
}
```

ODPHTTPErrorListenerDelegate protocol

Syntax

public protocol ODPHTTPErrorListenerDelegate

Remarks

Protocol for classes implementing listeners for HTTP error details. When the client connects to the reverse proxy or relay server through HTTP, during the course of establishing connection, there might be errors. The client can get notified of these errors on implementing this protocol.

*onHTTPError:errorMessage:httpHeaders:(int, NSString *, NSDictionary *) method*
Called when the ODPHTTPErrorListenerDelegate protocol is implemented by the class to receive the HTTP error details.

Syntax

`public virtual void onHTTPError:errorMessage:httpHeaders:(int` **code**`, NSString *` **message**`, NSDictionary *` **headers**`)`

Usage

```
The application has to implement the method to receive the error
message and the HTTP headers.
 Usage

 -(void)onHTTPError:(int)code errorMessage:(NSString*)message
httpHeaders:(NSDictionary*)headers
 {
     // Error handling using the error message and headers
 }
```

ODPPushDelegate protocol

Syntax

`public protocol ODPPushDelegate`

Remarks

Protocol for classes implementing push listener. All classes implementing the push delegate should use this protocol and this class should be passed to the delegate variable of the client connection class object.

*pushNotificationReceived:(NSDictionary *) method*
Notifies the listener with the data been pushed from the back-end.

Syntax

`public virtual void pushNotificationReceived: (NSDictionary *` **data**`)`

Usage

```
 Usage

 -(IBAction)buttonsPressed:(UIButton*)button{
     [ODPClientConnection registerForPayloadPush:self];
 }
```

```
                      -(void)pushNotificationReceived:data{
 {
     NSLog("Data : %", [data objectForKey:"Data"]);
 }
```

ODPUserManagerDelegate protocol

Syntax
`public protocol  ODPUserManagerDelegate`

Remarks
Protocol for classes implementing notification listeners. Protocol for classes implementing notification listeners. This class should be passed to the delegate variable of the user manager class object.

*userRegistrationFailed:userManager:(NSError *, id) method*
Called when the user registration has failed and the reason for failure is propogated through an error object.

Syntax
`public virtual void userRegistrationFailed:userManager:(NSError *` **error**`, id` **userManager**`)`

Usage

```
This is true in case of asynchronous user registration only.
 Usage

 -(IBAction)buttonsPressed:(UIButton*)button{
     ODPUserManager* userManager = [ODPUserManager
getInstance:"applicationid"];
    [userManager setConnectionProfileWithHost:@"hostname" port:5001
farm:@"farm" error:nil];
     [userManager setDelegate:self];
     [userManager registerUserWithUser:"username"
activationCode:@"act.code" error:nil flag:NO];
 }




                      -(void)userRegistrationFailed:(NSError*)error
 {
     // Error handling using the error object
 }
```

userRegistrationSuccessful() method

Called when the user registration is successful.

Syntax

```
public virtual void userRegistrationSuccessful ()
```

Usage

```
This is true in case of asynchronous user registration only.
 Usage

 -(IBAction)buttonsPressed:(UIButton*)button{
     ODPUserManager* userManager = [ODPUserManager
getInstance:"applicationid"];
    [userManager setConnectionProfileWithHost:@"hostname" port:5001
farm:@"farm" error:nil];
     [userManager setDelegate:self];
     [userManager registerUserWithUser:"username"
activationCode:@"act.code" error:nil flag:NO];
 }





                      -(void)userRegistrationSuccessful
 {
     // Successful post-processing
 }
```

CHAPTER 3 Developing Android Applications

Provides information about using advanced SAP Mobile Platform features to create applications for Android devices. The audience is advanced developers who are familiar working with APIs, but who may be new to SAP Mobile Platform.

Describes requirements for developing a device application for the platform. Also included are task flows for the development options, procedures for setting up the development environment and API references.

1. *Getting Started Task Flow for Android Applications*

   This task flow describes requirements for developing a device application for the platform. It includes task flows for the development options, procedures for setting up the development environment, and API documentation.
2. *Development Task Flow Using OData SDK (Messaging Channel)*

   After you import mobile applications and associated libraries into the Android development environment, use the Android API references to customize your device applications. For an application to work in an ODP scenario, it needs to be initialized first. Additionally, an application developer uses APIs specific to ODP that enables the application to send and receive data.
3. *Development Task Flow Using REST SDK (HTTP Channel)*

   The REST SDK library enables consumption of SAP Mobile Platform REST services with pure HTTPS connectivity.
4. *Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)*

   Use SDMConstants interface in SDMPreferences class to set the request-response between messaging channel or REST SDK.
5. *Deploying Applications to Devices*

   This section describes how to deploy customized mobile applications to devices.
6. *OData SDK Components and APIs*

   The Android OData SDK provides a set of features that help application developers build new applications on top of the Android platform. It supports the usage of the OData protocol with SAP additions (OData for SAP) and provides solutions for the most common use-cases an application developer meets with.

Getting Started Task Flow for Android Applications

This task flow describes requirements for developing a device application for the platform. It includes task flows for the development options, procedures for setting up the development environment, and API documentation.

See also

- *Development Task Flow Using OData SDK (Messaging Channel)* on page 177

Installing the Android Development Environment

Install the Android development environment, and prepare Android devices for authentication.

Installing the Android SDK

Install the Android SDK.

1. Confirm that your system meets the requirements at *http://developer.android.com/sdk/requirements.html*.
2. Download and install the supported version of the Android SDK starter package.

   See the *Google Android Versions* topic for your platform in *Supported Hardware and Software* at *http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories*. Select the appropriate version of the SAP Mobile Platform document set.
3. Launch the Android SDK Manager and install the Android tools (SDK Tools and SDK Platform-tools) and the Android API.
4. Launch the **Android Virtual Device Manager**, and create an Android virtual device to use as your emulator.

Installing ADT in SAP Mobile WorkSpace

Install the supported version of Android Development Tools (ADT) directly in the SAP Mobile WorkSpace Eclipse environment.

Follow the instructions for installing the ADT Plugin for Eclipse at *http://developer.android.com/sdk/installing/installing-adt.html*.

Setting Up the Development Environment

Set up the Android Development Environment by downloading the required plugins.

Prerequisites

- Download the Java Standard Edition (6 Update 24) Development Kit from the following URL: *http://www.oracle.com/technetwork/java/javase/downloads/index.html*
- Download Eclipse Helios (3.6.2) from the following URL: *http://www.eclipse.org/downloads/*

Task

1. Start the Eclipse environment.
2. From the **Help** menu, select **Install New Software**.
3. Click **Add**.
4. In the Add Repository dialog, enter a **Name** for the new plugin.
5. Enter one of the following for **URL**:
    - https://dl-ssl.google.com/android/eclipse/
    - http://dl-ssl.google.com/android/eclipse/
6. Click **OK**.
7. Select the **Developer Tools** checkbox and click **Next**.
8. Review the tools to be downloaded.
9. Click **Next**.
10. Read and accept the license agreement and click **Finish**.
11. Once the installation is complete, restart Eclipse.

Setting Up the Android SDK Library in the Plugin

Set up the Android SDK in the ADT Plugin.

1. In the Eclipse environment, from the Window menu, select **Preferences**.
2. In the left navigation pane, select the **Android** node.
3. Click **Browse** to search for the location where you have stored the Android SDK.
4. Click **Apply** and **OK**.

Adding User Permissions in Android Manifest File

Add user permissions to the Android project.

1. Open the Android manifest file in Eclipse project.

2. Add the permissions provided:

```
    <uses-permission
android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" ></uses-
permission>
```

Importing Libraries to your Android Application Project

Reference the libraries required for the Android application project.

1. Download the SDK/ODP library files to your host development system.
   - For Online Data Proxy, you need to download the following .jar files from the location: *SMP_HOME*`\MobileSDK<Version>\OData\Android\libraries\`:
     - `sup-client-util.jar`
     - `ClientLib.jar`
     - `SUPProxyClient*.jar`
   - For developing applications for native HTTP services, you can download the following .jar files from the location: *SMP_HOME*`\MobileSDK<Version>\OData\Android\libraries\Utils\`:
     - `DataVaultLib.jar`
     - `gcm.jar`
     - `perflib.jar`
     - `sap-e2etrace.jar`
2. Create a new folder, named `libs`, in your Eclipse/Android project.
3. Right click `libs` and choose **Import -> General -> File System**, then click **Next**.
4. Browse the file system to find the library's parent directory (where you downloaded it).
5. Click **OK**, then click the directory name (not the checkbox) in the left pane and check the relevant JAR in the right pane. This puts the library into your project (physically).
6. Right click on your project, choose **Build Path -> Configure Build Path**, then click the **Libraries** tab, then click **Add JARs...**
7. Navigate to your new JAR in the libs directory and add it. (This is when your new JAR is converted for use on Android.)

This procedure includes a Dalvik-converted JAR in your Android project and makes Java definitions available to Eclipse in order to find the third-party classes when compiling your project's source code.

Downloading the Latest Afaria Libraries

Afaria provides provisioning of configuration data and certificates for your SAP Mobile Platform client application. Afaria libraries are packaged with SAP Mobile Platform, but may not be the latest software available. To ensure you have the latest Afaria libraries, download Afaria software.

1. Navigate to the Mobile Enterprise Technical Support website at *http://frontline.sybase.com/support/downloads.aspx*.
2. If not registered, register for an account.
3. Log into your account.
4. Select **Software Updates** and download the latest Static Link Libraries.
5. Extract the contents of the downloaded zip file.
6. Copy the Afaria library files into the Android development environment.
7. Include the Afaria library into your project.

Development Task Flow Using OData SDK (Messaging Channel)

After you import mobile applications and associated libraries into the Android development environment, use the Android API references to customize your device applications. For an application to work in an ODP scenario, it needs to be initialized first. Additionally, an application developer uses APIs specific to ODP that enables the application to send and receive data.

This section provides a development task flow for creating Android applications that work in the ODP scenario.

1. *Initializing the Application*

   Initialize the application when it starts the first time.
2. *Setting Server Details*

   Set or update the connection properties of the server before or after user registration.
3. *Enabling Single and Mutual SSL Authentication*

   Secure sockets layer (SSL) is a protocol that governs certificate authentication. It exchanges the certificate information. SSL also encrypts all information that flows between a client and a server.
4. *Registering a User*

   Using a predefined authentication mechanism, register a user automatically.You can register the user either synchronously or asynchronously.
5. *Sending Data Request to the Back-end*
6. *Retrieving the Response from the Back-end*

   Based on the data request sent to the back-end, you need to retrieve the data as a response to the device.
7. *Using HTTPS over the SAP Mobile PlatformMessaging Channel*

(Optional) Clients can securely communicate with the backend via the server through the HTTPS transport protocol. By using SSL over the SAP Mobile Platform Messaging Channel, all messages are encrypted and authenticated to ensure secure communication.

8. *Debugging Runtime Error and Performance Analysis*

To handle occurrences of exceptions and special conditions that change the normal flow of the program execution, error handling has to be done appropriately.

See also

- *Getting Started Task Flow for Android Applications* on page 174
- *Development Task Flow Using REST SDK (HTTP Channel)* on page 196

Initializing the Application

Initialize the application when it starts the first time.

The following code illustrates how to initialize an application.

```
ODPUserManager.initInstance(getApplicationContext(), "TestApp");
ODPUserManager userManager=ODPUserManager.getInstance();
```

See also

- *Setting Server Details* on page 180

Downloading Customization Resource Bundles

(Optional) The application can download the custom resources using this API. The customized resource bundles are assigned to application connections in the SCC.

A customization resource bundle is a JAR file that includes a manifest file of name and version properties. It enables you to associate deployed client applications with different versions of customization resources. See *Application Customization Resource Bundles* in *SAP Control Center for SAP Mobile Platform* for information about the implementation task flow of customizations resource bundles.

When a notification is received for updates made to the CUSTOMIZATION_RESOURCE property, this API returns the customization resource bundle as binary data.

The following code illustrates how to download customization resource bundles.

```
ODPClientConnection.initInstance(getApplicationContext(), appName);
ODPClientConnection lm= ODPClientConnection.getInstance();
EndPointListner epl=new EndPointListener();
lm.addConfigurationChangeListener(epl);
serviceDocPath = las.getApplicationEndPoint();

public class EndPointListner implements
IODPConfigurationChangeListener{

      @Override
      public void configurationHasChanged(int key, String value) {
           Log.i("Configuration has changed", "key: "+arg0 + "Value:
```

```
"+arg1);
            String Custom_resource = "";
            if(key == CUSTOMIZATION_RESOURCES)
            {
                  ODPAppSettings las = new ODPAppSettings();
                  byte[] Property_metadata;
                  try {
                              Property_metadata =
las.getCustomizationResourceBundle(value,"supuser2","s3puser");
                  !
                  } catch (ODPException e) {
                  e.printStackTrace();
                  }
                  }
                  Log.i("Customization resources
test"+Custom_resource );
             }
       }
 }
```

Returns

When a notification is received for updates made to the CUSTOMIZATION_RESOURCE property, this API returns the customization resource bundle as binary data.

Registering Listeners for Push Notifications

(Optional) The application should register and implement a listener interface to receive native or payload push notifications.

To enable push notifications, include the `gcm.jar` available at *`SMP_HOME`*`\MobileSDK<Version>\OData\Android\libraries\` in your application project. For more information, refer to the information available in *http://developer.android.com/guide/google/gcm/gs.html* .

The `IODPPushNotificationListener` interface should be implemented by the application to receive the GCM native push notifications on Android devices. The `registerForNativePush` method in the `ODPClientConnection` class is used to register native push notifications. The `onGCMNotification` method is called when a new GCM push notification is received. For more information on configuring GCM native notifications on SCC, see *GCM Native Notification Properties* in *SAP Control Center for SAP Mobile Platform.*

Note: When the application is not running or terminated, the GCM notifications will not reach the client device. The IODPPushNotificationListener will not be invoked.

The following code illustrates how to register listeners for native push notifications.

```
public class MyPushListener implements IODPPushNotificationListener
{
public int onGCMNotification(Hashtable hashValues) {
// Utilize notification
log.i(“GCM Notification received”);
Return 0;
}
MyPushListener myPushListener = new MyPushListener();
ODPClientConnection.getInstance().registerForNativePush
(myPushListener);
}
```

The return values for onGCMNotification include:

- 0 – The client receives the payload notification from the server, if the **online/payload push with native notification** option is selected in the **SCC Push Configurations** tab. For more information, see *Configuring Native Notifications* in *SAP Control Center for SAP Mobile Platform*.
- 1 – The client does not receive the payload notification from the server.

To consume push messages, the application registers a listener object.The client SDK notifies this listener object whenever there is a push message from the server. The listener object should implement the ISDMNetListener interface.

The following code illustrates how to enable Android push notification on a device.

```
public class MyPushListener implements ISDMNetListener{
@Override
public void onError(ISDMRequest arg0, ISDMResponse arg1,
ISDMRequestStateElement arg2) {
Log.i("Error receiving push notification");
}

@Override
public void onSuccess(ISDMRequest arg0, ISDMResponse arg1) {
// TODO Auto-generated method stub
Log.i("Push notification received successfully");
}
}
MyPushListener myPushListener = new MyPushListener();
ODPClientConnection.getInstance().registerForPayloadPush(myPushList
ener);
}
```

Setting Server Details

Set or update the connection properties of the server before or after user registration.

The following code illustrates how to set the server details.

```
userManager.setConnectionProfile("relayserver.sybase.com",
5001,"sampleApp.FarmMBS");
```

See also

- *Initializing the Application* on page 178

Enabling Single and Mutual SSL Authentication

Secure sockets layer (SSL) is a protocol that governs certificate authentication. It exchanges the certificate information. SSL also encrypts all information that flows between a client and a server.

In a single SSL connection, the client must trust the server certificate. You can set this up in one of these ways:

- Install the CA certificate on the device trustStore that application uses while connecting to the HTTPS URL.
- If the application does not provide the certificate, implement the ISSLChallengeListener. The application receives a callback to the `isServerTrusted` method. When the application receives the server certificate, it can choose to trust the certificate or not. Based on that, it returns the Boolean value as true or false.

In mutual SSL connection, two parties authenticate each other through verifying the provided digital certificate (P12), so that both parties are assured of the others' identity.

The application must implement `IMutualSSLChallengeListener`, and invoke the `getClientCertificate` callback to request the client certificate. The client returns an X.509 certificate, and a private key, which together form an `HttpClientCertInfo` object.

Note:

- For mutual SSL, both single and mutual listeners need to be implemented.
- During onboarding with single and mutual SSL, add fully qualified domain name of the SAP Mobile Server. For example, vmw5541.wdf.sap.corp or vw<xxx>.dhcp.wdf.sap.corp.

Examples

- **Single SSL and Mutual SSL –** Implement the IODPCertificateChallengeListener and IODPMutualSSLChallengeListener for mutual SSL authentication. Following sample code uses Helper class which contains variables to implement single and mutual SSL.

```
public class Helper {

    public static String serverIP,Port,secCon,domain="default";
    public static String userName;
    public static String password;
    public static String serviceDocUrl;
    public static String companyID="";
    public static String
channel=SDMConstants.SDM_HTTP_HANDLER_CLASS;

    @SuppressLint("NewApi")
    public class IMO_Mutual_SSL_sync extends Activity implements
```

```
IODPCertificateChallengeListener, IODPMutualSSLChallengeListener{
    boolean reg1= false, reg2;

    @SuppressLint("NewApi")
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Registerapi();
    }

    public void Registerapi(){
        Helper.channel = SDMConstants.SDM_HTTP_HANDLER_CLASS;
            SDMLogger logger = new SDMLogger();
       ISDMPreferences pref = new SDMPreferences(getBaseContext(),
logger);
        try {
            Helper.channel = SDMConstants.SDM_IMO_HANDLER_CLASS;

pref.setStringPreference(ISDMPreferences.SDM_CONNECTIVITY_HANDLER
_CLASS_NAME, Helper.channel);

pref.setBooleanPreference(ISDMPreferences.SDM_PERSISTENCE_SECUREM
ODE, false);

        } catch (SDMPreferencesException e2) {
            // TODO Auto-generated catch block
            e2.printStackTrace();
            Log.i("tag","problem with preferences");
        }
        logger.setLogLevel(ISDMLogger.DEBUG|ISDMLogger.INFO);
        logger.logToAndroid(true);
        SDMConnectivityParameters param = new
SDMConnectivityParameters();
        param.setUserName(Helper.userName);
        param.setUserPassword(Helper.password);

        //start to get the certificate from the application
          SDMRequestManager requestManager = new
SDMRequestManager(logger, pref, param, 1);
          try {
            ODPUserManager.initInstance(getBaseContext(),
Helper.appId);
            ODPClientConnection lm = null;
            lm = ODPClientConnection.getInstance();
            ODPUserManager lum = null;

            lum = ODPUserManager.getInstance();

           try {

MessagingClientLib.getInstance().clearServerVerificationKey();
        } catch (MessagingClientException e) {

            e.printStackTrace();

            Log.i("tag", "Exception form Messaging client ");
        }

             try {
            lm.setODPMutualSSLChallengeListener(this);
            lm.setODPCertificateChallengeListener(this);
        } catch (ODPException e1) {
            // TODO Auto-generated catch block

            Log.i("tag", "in catch of
setODPCertificateChallengeListener");
            e1.printStackTrace();
        }catch (Exception e){
            Log.i("tag", "in Exception");
        }
        ODPUserManager.enableHTTPS(true);
        lum.setConnectionProfile(<host>, <port>), <farmID>);

        Helper.clientConn = new
ClientConnection(getApplicationContext(),
                Helper.appId, Helper.domain, Helper.secCon,
requestManager);
        Log.i("tag","executed init instance");

        Log.i("tag","in try block");

            lum.setConnectionProfile(Helper.serverIP
, Integer.parseInt(Helper.Port), Helper.companyID);

                if(!lum.isUserRegistered())
                {
                    Log.i("tag","registering user");

                 lum.registerUser(Helper.userName, Helper.secCon,
Helper.password, true);


                Helper.serviceDocUrl =
ODPAppSettings.getApplicationEndPoint();
                Log.i("tag", "Application end point" +
Helper.serviceDocUrl);
             Helper.pushendpt = ODPAppSettings.getPushEndPoint();
                Log.i("tag", "Push end point"+ Helper.pushendpt);
             Helper.channel = SDMConstants.SDM_IMO_HANDLER_CLASS;

                Intent s=new
Intent(IMO_Mutual_SSL_sync.this,RR_SettingsExchange.class);
                startActivity(s);

            }
            } catch (ODPException e2) {
                // TODO Auto-generated catch block
                Log.i("tag", "Registration failed error code" +
e2.getErrorCode() + "error message" + e2.getMessage());
                e2.printStackTrace();
            }
        }
```

```
//mutual ssl
// start read from application
    @Override
    public ClientCertificateInformation getClientCertificate() {
        // TODO Auto-generated method stub

        InputStream inStream;

        Log.i("tag", "in getclientcertificate");

        try {
               // get the p12 certificate as an inputstream
             int dd = R.raw.supuser; //location of p12 file in the
application
               Context context = this.getApplicationContext();
               inStream =
context.getResources().openRawResource(dd);

            //  Create a KeyStore,load it with the inputstream,get
the alias
               KeyStore ks = KeyStore.getInstance("PKCS12");
               ks.load(inStream, "mobile".toCharArray());
               Enumeration aliases = ks.aliases();
               String keyname = (String) aliases.nextElement();

             // Using the alias retrieve the certificate n private
key
               // cast them into X509Certificate and PrivateKey
respectively
               PrivateKey pk = (PrivateKey) ks.getKey(keyname,
                            "mobile".toCharArray());
               X509Certificate xcer = (X509Certificate)
ks.getCertificate(keyname);

               return new ClientCertificateInformation(xcer, pk);

        } catch (Exception e) {

            Log.i("tag", e.getMessage());

            return null;
        }


    }

    @Override
    public boolean isServerTrusted(ODPCertInfo[] arg0) {
        // TODO Auto-generated method stub
        Log.i("tag", "In Is server trusted");
        return true;
    }


//end read from application
```

```
}
```

Registering a User

Using a predefined authentication mechanism, register a user automatically.You can register the user either synchronously or asynchronously.

The following code illustrates how to automatically register a user.

```
userManager.registerUser("supuser","SSO2Cookie","s3puser", false);
```

Sending Data Request to the Back-end

See also

- *Retrieving the Response from the Back-end* on page 186

Creating a URL Request

Create a URL request that enables the device to forward a data request to the corresponding back-end.

The following code illustrates how to create a URL request.

```
SDMBaseRequest getrequest = new SDMBaseRequest();
String serviceDocUrl = ODPAppSettings.getApplicationEndPoint();
getrequest.setRequestUrl(serviceDocUrl);
```

Enabling XCSRF Token Protection

XCSRF token can be enabled to secure the backend data.

Once you have enabled XCSRF token on the device, the token is extracted after the first GET request triggered by the application. This token is retained in the device memory, and is added to all subsequent modifying requests.

The following code illustrates how to enable XCSRF token.

```
 protected SDMConnectivityParameters param = new
        SDMConnectivityParameters();
        //if(Application end point is a Odata UrL) then
        param.enableXsrf(true);
```

Assigning Credentials

Assign the user credentials for Gateway.

The following code illustrates how to assign the user credentials for Gateway.

```
SDMConnectivityParameters param = new SDMConnectivityParameters();
param.setUserName("supuser123");
param.setUserPassword("123");
```

Adding Custom Headers

Add custom headers to a request message. This is a name/value pair that defines the operating parameters of an HTTP transaction. Custom headers are optional while sending a data request to the back-end.

The following code illustrates how to add custom headers to the request message.

```
Map<String,String>cookie = null;
cookie.put("KKNBJDNADU23123SHSFH", "cookie");
getRequest.setHeaders(cookie);
```

Setting the Required Timeout

Set the timeout value up to which the application waits until the request reaches the server.

The following code illustrates how to set the required timeout.

```
SDMLogger logger = new SDMLogger();
SDMPreferences pref = new SDMPreferences(mContext, logger);
pref.setIntPreference(ISDMPreference.SDM_CONNECTIVITY_CONNTIMEOUT,
7000);
```

Forwarding the Request

Send the asynchronous request message to the back-end

The following code illustrates how to forward an asynchronous message.

```
SDMRequestManager reqManager = new SDMRequestManager(logger, pref,
param, 1);
reqManager.makeRequest(getRequest);
```

Retrieving the Response from the Back-end

Based on the data request sent to the back-end, you need to retrieve the data as a response to the device.

The following code illustrates how to retrieve the response from the back-end to the device

```
public void onSuccess(ISDMRequest aRequest, ISDMResponse
aResponse)
{
 String serviceDoc = EntityUtils.toString(aResponse.getEntity());
}
```

See also

- *Sending Data Request to the Back-end* on page 185

Using HTTPS over the SAP Mobile PlatformMessaging Channel

(Optional) Clients can securely communicate with the backend via the server through the HTTPS transport protocol. By using SSL over the SAP Mobile Platform Messaging Channel, all messages are encrypted and authenticated to ensure secure communication.

See also

- *Debugging Runtime Error and Performance Analysis* on page 190

Enabling HTTPS as a Transport Protocol

You can set HTTPS as the transport protocol that the OData client should use to communicate with any host (Example: Relay server).

Syntax

```
public static void enableHTTPS(boolean useHTTPS) throws ODPException
```

Parameters

- **useHTTPS –** Set to "true", if the protocol to be used is HTTPS.

  Set to "false", if the protocol to be used is HTTP.

  Note: The default protocol is HTTP.

Enabling a Listener for HTTPS Support with Server Certificate Validation

When a client attempts to connect to a host (like a relay server), this connection is authenticated with a basic authentication challenge.To verify if the server certificate is trusted or not, the application registers a listener object with the OData SDK. If the server certificate is trusted, the connection is successful. If the server certificate is not trusted, it is rejected and the connection fails.

See *Configuring SAP Mobile Server to use Relay Server* in *SAP Control Center for SAP Mobile Platform* for information on the configuration of relay server properties.

Syntax

```
public void
setODPCertificateChallengeListener(IODPCertificateChallengeListener
oListener) throws ODPException;
```

Parameters

- **oListener –** Listener object that implements ODPClientListeners.IODPCertificateChallengeListener interface.

Examples

- **Implement the listener**

```
Public class ODPCertificateListener implements
IODPCertificateChallengeListener
{
    // callback method for certificate verification
    @Override
   public boolean isServerTrusted(ODPCertInfo[] certCredentials)
```

```
    {
        Log.i("Do you trust the server for
credentials:"+certCredentials.toString()); //display these
credentials for the server
       return true; // return true or false depending on userinput
from the UI
    }

}
```

- **Register a listener to verify server certificate**

```
ODPCertificateListener odpCertificateListener = new
ODPCertificateListener();
ODPUserManager.getInstance().setODPCertificateChallengeListener(o
dpCertificateListener);
```

Enabling a Listener for HTTPS Support with Basic AuthChallenge

When a client attempts to connect to a host (like a relay server), this connection is authenticated with a basic authentication challenge.To setup a basic authentication, the application registers a listener with OData SDK. If the IODPHTTPAuthChallengeListener is not registered, an HTTP_AUTH_FAILURE error is returned when a challenge is received.

Syntax

```
public static void
setODPHTTPAuthChallengeListener(ODPClientListeners.IODPHTTPAuthChal
lengeListener listener) throws MessagingClientException;
```

Parameters

- **listener –** Listener object that implements the ODPClientListeners.IODPHTTPAuthChallengeListener. The object invokes the callback method of this interface when a connection receives the HTTP_AUTH_FAILURE (HTTP response code 401) challenge

Examples

- **Register a Listener for HTTPS Auth Challenge**

```
ODPCredentials odpListener = new ODPCredentials();
ODPClientConnection.setODPHTTPAuthChallengeListener(odpListener);
```

- **Implement the listener**

```

public class UserRegistration{

     public void startUserRegistration() {
       ODPUserManager.initialize(appID);
       ODPUserManager.setConnectionProfile(serverIP, serverPort,
farmID);
```

```
       ODPUserManager.enableHTTPS(true);
       ODPCredentials odpListener = new ODPCredentials();

ODPClientConnection.setODPHTTPAuthChallengeListener(odpListener);
       ODPUserManager.registerUser(username, securityConfig,
password);
     }

final static class ODPCredentials implements
IODPHTTPAuthChallengeListener
{
    // callback method for HTTP authentication 401 challenge
    @Override
    public ODPHTTPAuthChallengeCredentials getCredentials(String
sHostName, String sOldUserName, String sRealm)
     {
        // TODO Auto-generated method stub
        Log.i("Get Credentials for host"+sHostName);
        ODPHTTPAuthChallengeCredentials odpCredentials = new
ODPHTTPAuthChallengeCredentials(userName,pwd);
        return odpCredentials;
    }
}
}
```

Registering a Listener for HTTP Error in Android Applications

To ensure that OData Android clients are notified of HTTP errors while establishing a connection with the network edge, implement a listener.

Syntax

```
public static void setODPHTTPErrorListener(IODPHttpErrorListener
oListener) throws MessagingClientException
```

Parameters

- **oListener –** listener object that implements the interface `IODPHttpErrorListener`.

Examples

- **Implement the listener**

```

public class UserRegistration{

     public void startUserRegistration() {
       UserManager.initialize(appID);
       UserManager.setConnectionProfile(serverIP, serverPort,
farmID);
       UserManager.enableHTTPS(true);
      ODPErrorListener odpErrorListener = new ODPErrorListener();

ODPClientConnection.setODPHTTPErrorListener(odpErrorListener);
```

```
       UserManager.registerUser(username, securityConfig, 
password);
     }

public class ODPErrorListener implements IODPHttpErrorListener 
{
    // callback method for HTTP Error Code Listener
    @Override
    public void onHttpError(int iErrorCode, String sErrorMessage, 
Hashtable oHeaders)
     {
        // TODO Auto-generated method stub
        Log.i("Error info" +iErrorCode+sErrorMessage);
      }
}
```

Debugging Runtime Error and Performance Analysis

To handle occurrences of exceptions and special conditions that change the normal flow of the program execution, error handling has to be done appropriately.

The following code illustrates how to handle errors for asynchronous requests.

```
public void onError(ISDMRequest aRequest, ISDMResponse aResponse, 
ISDMRequestStateElement aRequestStateElement)
{
Boolean verdict = false;
}
```

See also

- *Using HTTPS over the SAP Mobile PlatformMessaging Channel* on page 186

End to End Tracing

(Optional) End to end tracing enables an application developer or end user to trace a request that is sent from the client to the back-end. This spans the entire landscape where you can derive a correlation of traces at the client, server and back-end.

These correlated traces help in performance analysis and are centrally monitored on SAP Solution Manager. These are displayed as reports where you can extract information on failure of delivering a request, time taken for a request to reach a component and so on.

On the client side, the client framework enables an application developer to switch on the trace for messages. The client traces the request at predefined points and all these transactions/ requests are recorded in a Business Transaction XML. Additionally, the client maintains a unique identifier in the HTTP header called the SAP Passport that is used to correlate traces across various components. This Business Transaction XML can later be uploaded to the SAP Solution Manager which is a central location to correlate all logging information.

The necessary condition for any client application to use the end to end tracing feature is that the application should be registered with SAP Mobile Platform through the related registration mechanism as applicable for the respective device platform to acquire an application connection ID. This application connection ID should be included in every communication with the server. This application connection ID is also used by the administrator to enable end to end tracing for the respective client application.

Using Tracing APIs in SUPProxyClient Library

You can perform end to end tracing by using supportability APIs defined in the SUPProxyClient library.

This approach is recommended for applications which use SAP Mobile Platform client framework to consume the platform services. These APIs deal with the internal complexities and necessities of the end to end tracing configuration and provide simple and easy to use interface for the application developer. These APIs are available in the `ODPClientConnection` class.

Syntax

- Use the `setTraceLevel` API to set the passport trace level. The trace level can be set to ODPClientConnection.TrcLvl_HIGH , ODPClientConnection.TrcLvl_MEDIUM ,ODPClientConnection.TrcLvl_LOW, or, ODPClientConnection.TrcLvl_NONE .

  ```
  public void setTraceLevel(int level)
  ```
- Use the `startTrace` API to check if end to end tracing is enabled and thereafter start tracing the message.

  ```
  public void startTrace()
  ```
- Use the `stopTrace` API to disable end to end tracing and thereafter stop tracing the message.

  ```
  public void stopTrace()
  ```
- Use the `uploadTrace` API to forward the generated business transaction XML to the server.

  ```
  public void uploadTrace()
  ```

Note: For more information on how to configure the SAP Solution Manager, see *Configuring SAP Solution Manager URL* in *SAP Control Center for SAP Mobile Platform*.

Examples

- **Setting the Trace Level -**

  ```
  ODPClientConnection ocl = ODPClientConnection.getInstance();
  ocl.setTraceLevel(ocl.TrcLvl_HIGH);
  ```
- **Starting a Trace -**

  ```
  ocl.startTrace();
  ```
- **Stopping a Trace -**

  ```
  ocl.stopTrace();
  ```

- **Uploading a Business Transaction XML to SAP Solution Manager –**

  ```
  ocl.uploadTrace();
  ```

Using Tracing APIs in Standalone SUPSupportability Library

You can perform end to end tracing by using standalone SUPSupportability library.

This approach is recommended for the applications which do not use SAP Mobile Platform client framework but still consume the platform services. You can configure the client for end to end tracing by:

- Creating a Passport:

  A passport is a string of encoded parameter values used in controlling logging information and taking logging decisions across various components in server and client runtime. The passport is used to log debugging information as directed by the passport configuration.

Table 4. Components of Passport

Component	Description
Root Context Id	A unique ID that represents a client/application. This ID should be same for all generated passports in a tracing session.
Transaction Id	An ID that represents a transaction (one cycle of request-response) performed by the client or application. This is useful for tracking a request end-to-end. A new transaction ID should be created for each new request.
Trace Level	The trace level controls what and how much should be logged. The trace level can be set to HIGH, MEDIUM, or LOW. **Note:** The trace level is always specified by the administrator.

Note: It is mandatory for the client application to include a passport with communication to the server and each and every component in the server execution path.

- Business Transaction XML Creation:

  The Business Transaction XML (BTX) contains the client side trace which includes the client-server communication trace (request-response), as well as, the other trace generated by various execution points/entities in the application. The BTX creation starts when the application starts a tracing session and stops when the trace session is ended.

Usage

Follow the instructions to enable end to end tracing using the standalone SUPSupportability library. Once you have the traces loaded into a Business Transaction XML, you can upload to SAP Solution Manager.

1. Enable end to end tracing
   The following code illustrates how to start the Business Transaction XML creation.

```
//Get a handle to the BTX writer and do required initialization
steps
 com.sap.smd.e2e.trace.bustrans.impl.TraceController
traceController =
com.sap.smd.e2e.trace.bustrans.impl.TraceController.getInstance()
;

//set the trace level we are going to use for the passport. could
be obtained as passport.getTraceFlag() if the passport is already
available

 traceController.setTraceFlag(<trace level int value>);

//set the deviceId which will be used for identifying the BTX
specific to a device.
 traceController.setDeviceId(deviceId);

//Inform the trace controller to start the trace session and
initialize BTX creation. Specify the application name/id as BTX
name. appName is used to identify an application specific BTX on
the Solution Manager.

 traceController.startTransactionResult(appName);

Note:
‘appName’ is the SAP Mobile Platform Application Id.
‘deviceId’ can be obtained from the http response (to the device
registration request) cookie with key as ‘X-SUP-
 APPCID’.
--------------------------------------------------------------------
-----------
```

2. Trigger end to end trace, create an instance of SAP Passport.

```
com.sap.smd.e2e.trace.passport.IPassport passport =
com.sap.smd.e2e.trace.passport.PassportFactory.createPassport();

com.sap.smd.e2e.trace.passport.IGuid transactionId =
com.sap.smd.e2e.trace.passport.GuidGenerator.generateBusinessTran
sactionId();
passport.setTransactionId(transactionId);

passport.setRootContextId(traceController.getRootContextId());

passport.setTraceFlag(com.sap.smd.e2e.trace.passport.DsrUtils.Tra
ceFlags.TrcLvl_LOW);  //similarly TrcLvl_MEDIUM and TrcLvl_HIGH
could be used as required.
```

```
//Get the encoded passport string
String encodedPassport = passport.getEncoded();
```

Note: Create a new passport for every new request that the application makes to the server. Add the encoded passport string to the request headers with key as SAP-PASSPORT. This is required to uniquely identify or track a request end to end.

3. Add the request-response traces to the Business Transaction XML.

```
 //Create a transaction message using the various request-response
parameters
and pass it to the BTX writer

com.sap.smd.e2e.trace.bustrans.IRequest request = new
com.sap.smd.e2e.trace.bustrans.impl.Request();

//SAP-PASSPORT and X-SUP-APPCID headers need to be set into the
trace request along with other headers. SAP-PASSPORT will be the
same  passport used in the request headers (to server). X-SUP-
APPCID is the device ID or application connection ID
available in device registration response cookie.
request.setHeaders(<Request Headers>);
request.setContent(<Request Line>);
request.setSentBytes(<number of bytes sent>);

com.sap.smd.e2e.trace.bustrans.IResponse response = new
com.sap.smd.e2e.trace.bustrans.impl.Response();
response.setHeaders(<Response Headers>);
response.setContent(<Response Data>);
response.setResponseCode(<response code>);
response.setReceivedBytes(<number of bytes received>);

traceController.createStep(request);

//Note: The following is a mandatory and important step which ties
a transaction step(request) with a transaction ID.
//Since we create a new passport for a new request and set a
transaction ID for the passport every time, it is good to use the
same
transaction ID here for better correlation

traceController.addTransactionId(request,
passport.getTransactionId().getHex()); //the transaction id used
in the passport generated for current request.

//Add these collected transaction (request-response details/
statistics) to the BTX document
traceController.addMessage(request, response, firstByteSent,
lastByteRcvd);
```

4. Stop or disable end to end tracing

   The following code illustrates how to stop the Business Transaction XML creation.

```
traceController.stopTransactionResult();
```

5. Get the generated Business Transaction XML.

```
byte[] btx = traceController.getTransactionXML();
// Client application can save this byte array into a file that can
be uploaded
```

6. Upload Business Transaction XML to Solution Manager.

   Note: For more information, see the *Uploading Business Transactions for Tracing* in *Developer Guide: REST API Applications*.

Analyzing Performance Data Points

(Optional) To analyze the performance of the client, measurement points are available at different stages in a request-response cycle. These points are used to provide logs that help in assessing the processing time across various components in the SAP Mobile Platform environment.

Note: The response time will be impacted after importing performance library. If you set the custom settings in SAP Control Center as true for logs, then there will be a degradation in the response time.

Data Points for the Client

The table below provides the list of data points and the log readings across which the performance can be measured.

Log Reading	Application	Proxy Client	Messaging Client	Network (includes reverse proxy/ relay server)	SAP Mobile Platform Server	Network	Enterprise Information System (EIS)
E2E: RR		X	X	X	X	X	X
ODP:RR			X	X	X	X	X
IMO:RR			X	X	X	X	X
Net-work:RR				X	X	X	X

Performance Readings

The above log readings determine the time elapsed across various stages of a request response (RR) cycle.

- E2E:RR - Corresponds to the performance reading when the request is forwarded from the proxy client and the response reaches the proxy client.
- ODP:RR - Corresponds to the performance reading when the request reaches the Messaging Client and the response reaches the Messaging Client.

- IMO:RR - Corresponds to the performance reading when the request is forwarded from the Messaging Client and the response reaches the Messaging Client.
- Network:RR - Corresponds to the performance reading when the request is forwarded from the network and the response reaches the network.

The above log readings determine the time elapsed across various stages of a request response (RR) cycle. Use these readings to determine processing time across the following components:

- Time taken at the messaging client: **ODP:RR** - **IMO:RR**
- Time taken at the proxy client: **E2E:RR** - **ODP:RR**

Development Task Flow Using REST SDK (HTTP Channel)

The REST SDK library enables consumption of SAP Mobile Platform REST services with pure HTTPS connectivity.

This section provides a development task flow for creating Android applications using REST Client library.

See also

- *Development Task Flow Using OData SDK (Messaging Channel)* on page 177
- *Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)* on page 227

Creating and Initializing the Client Connection

Create and initialize SMPClientConnection.

The `SMPClientConnection` class declares the programmatic interface for an object that manages connection settings required for registering the user and fetching application settings from the server. An `SMPClientConnection` object has to be initialized and connection properties has to be set on this object before performing any type of user onboarding or before fetching any application settings from the server.

Syntax

The following code illustrates how to initialize an application:

```
 public ClientConnection(Context context, String appID, String
Domain, String securityConfig, SDMRequestManager requestManager)
```

Examples

- **Example to initialize the client connection –** ClientConnection is required to set the parameters required for the Registration/Request-Response with mobile platform.

```
public static SDMLogger logger;
public static SDMPreferences pref;
public static SDMConnectivityParameters param;
public static SDMRequestManager reqMan;
logger = new SDMLogger();
pref = new SDMPreferences(c, logger);
pref.setBooleanPreference(ISDMPreferences.SDM_PERSISTENCE_SECUREM
ODE, false);
pref.setIntPreference(ISDMPreferences.SDM_CONNECTIVITY_HTTPS_PORT
, 443);
pref.setIntPreference(ISDMPreferences.SDM_CONNECTIVITY_CONNTIMEOU
T, 70000);
pref.setIntPreference(ISDMPreferences.SDM_CONNECTIVITY_SCONNTIMEO
UT, 70000);
param = new SDMConnectivityParameters();
param.setUserName(<username>);
param.setUserPassword(<password>);
reqMan = new SDMRequestManager(logger, pref, param, 1);

ClientConnection clientConnection = new
ClientConnection(getApplicationContext(), “NewFlight”, “default”,
“SSO”, reqMan);
```

Usage

If the `SDMRequestManager` object defined above is used for Request/Response, set the following additional preference:

```
pref.setStringPreference(ISDMPreferences.SDM_CONNECTIVITY_HANDLER_C
LASS_ NAME, SDMConstants.SDM_HTTP_HANDLER_CLASS);
```

Enabling Network Edge for HTTP

Set up your HTTP client to use the siteminder authentication type.

SAP Mobile Platform REST Client APIs supporting siteminder within SDMRequestManager class.

Examples

- **Example for registering a listener in**

```
public class TestScenario implements
IAuthenticationChallengeListner
{
private void Testscenario()
{
//Register for the Listener

SDMRequestManager.setAuthentationChallengeListener(this);
/** Perform registration, do not provide login credentials to
perform registration here, credentials to be provided on call
back  */

}
//call back method to return basic authentication credentials
@Override
public ICredential OnAuthenticationFailed(Authenticationschema
authScheme,HttpResponse response) {
        String test =authScheme.getRealm();
        String schema = authScheme.getSchemeName();
        BasicCredetials credentials = null;
        if(schema == "Basic"){
        credentials = new BasicCredetials("<username>",
"<password>");

        }
        return  credentials;
        //else // implement your own Icredentials listener

    }
}
```

Enabling Single and Mutual SSL Authentication

Secure sockets layer (SSL) is a protocol that governs certificate authentication. It exchanges the certificate information. SSL also encrypts all information that flows between a client and a server.

In single SSL connection, the client needs to trust the server certificate. This can be done one of the three ways:

- The CA certificate can be installed on the device trust store which the application takes while connecting to the HTTPS URL.
- The CA certificate can be bundled in the application and the `setServerCertificate` method can be invoked to provide the certificate. This is not a recommended way, but can used in case of backward compatibility.

  Note: Applicable to android device older than 4.0 version.
- If the application does not provide the certificate, application can implement the `ISSLChallengeListener`. The application receives a callback to the `isServerTrusted` method. Where the application receives the server certificate it can choose to trust the certificate or not. Based on that it returns the boolean value as true or false.

In mutual SSL connection, the server needs to trust the client certificate. In this case the application implements `IMutualSSLChallengeListener`. And `getClientCertificate` callback method is invoked to request the client certificate. The client returns an X.509 certificate, and a private key, which together will be formed into an `HttpClientCertInfo` object.

Note:

- For mutual SSL, both single and mutual listeners need to be implemented.

- During onboarding with single and mutual SSL, add fully qualified domain name of the SAP Mobile Server instead of providing the server IP address. For example the fully qualified domain name: vmw5541.wdf.sap.corp or vw<xxx>.dhcp.wdf.sap.corp.

Examples

- **Single SSL and Mutual SSL –** Implement listeners for SSL callbacks.

```
public class LaunchActivity extends Activity implements
ISSLChallengeListener, IMutualSSLChallengeListener {


    SDMRequestManager srm;
    ISDMLogger logger;
    ISDMConnectivitiyParameters params;
    ISDMPreferences prefer;
    String url="http://vmw3815.wdf.sap.corp:50009/sap/opu/sdata/
iwfnd/RMTSAMPLEFLIGHT/";

    public void mreg(View v){
        ClientConnection clientConnection;
        UserManager userManager;
        logger=new SDMLogger();
        prefer=new SDMPreferences(this, logger);
        params=new SDMConnectivityParameters();
        params.setLanguage("en");
        params.setUserName("supuser2");
        params.setUserPassword("s3puser");

        Context c = getApplicationContext();
          try{ InputStream certStream =
c.getResources().openRawResource(<certificate>);
CertificateFactory cf = CertificateFactory.getInstance("X509");
 Certificate cer= cf.generateCertificate(certStream);


         param.setServerCertificate(cer); //to place certificate
in the devices less than 4.0

          Log.i("tag", "In Set Server certificate API");

          }catch(Exception e){
              Log.i("tag", "set certificate from file failed" +
e.getMessage());
              e.printStackTrace();
          }


        srm=new SDMRequestManager(logger,prefer,params,1);

        /**
        * The set methods which should be called to register the
listeners to the request manager
         *
         */
        srm.setSSLChallengeListener(this);
        srm.setMutualSSLChallengeListener(this);


        ClientConnection.initInstance(getApplicationContext(),
"<application>", "<domain>", "<security configuration>", <request
manager>);
        try {
            j=ClientConnection.getInstace();
            j.setConnectionProfile("<connection url>");
            UserManager.registerUser(true);



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

    }
```

Send X.509 certificate and private key to the client as an HttpClientCertInfo object.

```
public HttpsClientCertInfo getClientCertificate()
{
        InputStream inStream;
        int cert="<p12 certificate location>";
         Context context=this.getApplicationContext();
         try {


             inStream =
context.getResources().openRawResource(cert);

             /*following code loads the key store, and retrieves
               The alias under which it is loaded. */

             KeyStore ks = KeyStore.getInstance("PKCS12");
                ks.load(inStream, "mobile".toCharArray());
                Enumeration aliases=ks.aliases();
                String keyname=(String)aliases.nextElement();

             /* using the alias stored in the variable “keyname”
            We get the Private Key and the X509  Certificate
            Which we use to return HttpClientCertInfo object.*/

             PrivateKey
pk=(PrivateKey)ks.getKey(keyname,"mobile".toCharArray());
             X509Certificate xcer=(X509Certificate)
(ks.getCertificate(keyname));


             return new HttpsClientCertInfo(xcer,pk);}
         catch(Exception e){
             return null;
         }



    }
/*
     * This is the callback which is invoked if
     * 1) SSL is enabled,
    * 2) The correct certificate is not installed on the device,in
which case it   is required of the application to verify the server
certificate.
     */

@Override
    public boolean isServerTrusted(X509Certificate[]
paramArrayOfX509Certificate) {
        // TODO Auto-generated method stub

        return true;
    }

}
```

Registering the User

Registers the user to the SAP Mobile Platform on-premise/cloud.

You need to send username and password as part of request manager during ClientConnection initialization. **isSynchronous** specifies whether this is a synchronous or an asynchronous registration. If asynchronous, then you have to implement and register 'IODPUserRegistrationListener' to get a call back.

Syntax

```
public boolean registerUser(boolean isSynchronous) throws
SMPException
```

Examples

- **Example for registering the user**

```

      For synchronous  registration
      Try{
        UserManager userManager = new
UserManager(clientConnection);
        userManager.registerUser(true);
        }
        Catch(SMPException e){
              e.printStackTrace();
       }


       For Asynchronous registration
       Try{
       UserManager userManager = new
UserManager(clientConnection);
       userManager.setUserRegistrationListener(this);
       userManager.registerUser(false);
       }
       Catch(SMPException e){
              e.printStackTrace();
       }

          @Override
           public void onAsyncRegistrationResult(State
registrationState, ClientConnection conn, int errCode, String
errMsg) {
           if(registrationState == State.SUCCESS)
           {
                 Log.i("RegistrationState
Success",conn.getServerHost()+conn.getApplicationID()
+conn.getDomain()+conn.getRequestManager().toString());

           }
           if(registrationState == State.FAILURE)
           {
                Log.i("Registration State Failure",errCode+errMsg
+conn.getServerHost());
           }
     }
```

Enabling CAPTCHA

Enable CAPTCHA during registration.

Sets the CAPTCHA Challenge Listener in case of captcha enabled by the SMP cloud. User has to implement the 'ISMPCaptchaChallengeListener' and set it using the above listener to get the captcha call back.

Syntax

Use `setCaptchaChallengeListener` to enable CAPTCHA challenge for registration.

```
public void setCaptchaChallengeListener
(ISMPCaptchaChallengeListener captchaListener)
```

Returns

The callback method 'validateCaptcha' has to return the captcha text for registering the user. Alternately, it can also return 'null' in which case, the captcha text has to be stored by the application and a subsequent registerUser(captchaText, isSynchronous) has to be explicitly called.

Examples

- **Example to enable CAPTCHA challenge**

  ```
  public class LoginActivity extends Activity implements
  ISMPCaptchaChallengeListener{
  //application logic
  clientConnection = new
  ClientConnection(getBaseContext(),<application
  id>,<domain>,<security config>,<requestManager>);
  UserManager userManager = new UserManager(clientConnection);
  userManager.setCapthaChallengeListener(this); // this class
  implements ISMPCaptchaChallengeListener
  Public String validateCaptcha(String captcha){
  //show captcha  and get captchaText
  Return captchaText;
  }
  }
  ```

Exchanging Settings between Client and Server

Settings exchange between the client and the SAP Mobile Server using REST SDK.

`getConfigPropertyMap` method throws SMPException and sets the value(s) of modifiable properties sent as HashMap of Key-Value pairs to be updated. `getConfigPropertyMap` method returns all the properties key-value pair.

Syntax

```
public Object getConfigProperty(String key) throws SMPException;
```

Returns a HashMap<String, String> of Key-Value pairs of Properties.

```
public HashMap<String,String> getConfigPropertyMap () throws
SMPException
```

Sets the value(s) of updatable properties sent as HashMap of Key-Value pairs to be updated.

```
public void setConfigProperty(HashMap<String,String> list) throws
SMPException
```

Examples

- **Example to get the configuration property**

  ```
  String RegistrationId =
  appSettings.getConfigProperty(“d:AndroidGcmRegistrationId");
  ```

  ```
  HashMap<String,String> properties =
  appSettings.getConfigPropertyMap();
  ```

- **Example to set configuration property**

  ```
  HashMap<String,String> properties = new HashMap<String,String>();
  properties.put("d:AndroidGcmRegistrationId", "1234565");
  appSettings.setConfigProperty(properties);
  ```

Downloading Customization Resource Bundles

(Optional) Download customization resource bundles as defined in SAP Mobile Platform.

Syntax

```
public byte[] getCustomizationResourceBundle(String CustomResource)
throws SMPException
```

Examples

- **Example for downloading customization resource bundles**

  ```
  byte[] custRes =
  appSettings.getCustomizationResourceBundle("<resourcebundlename>:
  <version>");
  ```

Getting an Application Endpoint

Retrieve the application endpoint URL.

Returns the value of Application Endpoint. It is used to perform request-response with the gateway.

Syntax

```
public String getApplicationEndPoint() throws SMPException
```

Examples

- **Example to get application endpoint URL**

  ```
  String DocPath = appSettings.getApplicationEndPoint();
  ```

Getting Push Endpoint

Retrieving push endpoint URL.

Returns the value of push endpoint. It is used to register for push notification.

Syntax

```
public String getPushEndPoint() throws SMPException;
```

Examples

- **Example to retrieve push endpoint URL**

  ```
  String PushEndPoint = appSettings.getPushEndPoint();
  ```

Setting Password Policy Using DataVault

Returns the DVPasswordPolicy (as defined in `com.sybase.persistence.DataVault`) which holds the information of the Password Policy.

Note: (Applicable to On-Premise only) To make the password policy editable in the SAP Mobile Server, set the application connection property "CapabilitiesPasswordPolicy" to true in the client application.

Syntax

```
public com.sybase.persistence.DataVault.DVPasswordPolicy
getPasswordPolicy() throws SMPException
```

Examples

- **Example for datavault password policy**

  ```
  DVPasswordPolicy passwordPolicy =
  appSettings.getPasswordPolicy();
  ```

Subscribing for GCM Push Notifications

(Optional) Subscribe to get GCM push notifications.

To enable GCM push notification for an Android application:

1. See *Enabling the GCM Service*.
2. Configure SCC for server side configuration to receive GCM push notification. For more information on configuring native notification, see *Configuring Native Notifications* in *SAP Control Center for SAP Mobile Platform*.

You can obtain the sender ID from the client application configured in SCC by invoking `getConfigProperty`() with the property name `d:AndroidGcmSenderId`. Using this sender ID, you should register `GCMRegistrar.register("<sender ID>")`.

Note:

- On-premise - Obtain the sender ID from the property `d:AndroidGcmSenderId`.
- On cloud - Provide the sender ID from the application directly.

Syntax

In the `onRegistered` callback of `GCMBaseIntentService`, set the GCM registeration ID obtained from the google to the property `d:AndroidGcmRegistrationId`.

```

        @Override
```

```
       protected void onRegistered(Context arg0, String
registrationId)
       {
        try
         {
            public static AppSettings appSettings = new
AppSettings(clientConnection);
           HashMap<String, String> fullConfig = new HashMap<String,
String>();
            fullConfig.put("d:AndroidGcmRegistrationId",
registrationId);
            appSettings.setConfigProperty(fullConfig);
         }
        catch(SMPException e)
        {
            Log.i("tag", "Error from update settings " +
e.getMessage() + "Error code" + e.getErrorCode());
            e.printStackTrace();
        }
.    }
```

Usage

Notifications can be sent to the device for the push endpoint obtained from the `getPushEndPoint` (). Notifications sent will be received to `onMessage` call back of `GCMBaseIntentService` ().

End to End Tracing

(Optional) End to end tracing enables an application developer and end user to trace a request that is sent from the client to the back-end. This spans the entire landscape where you can derive a correlation of traces at the client, server and back-end.

These correlated traces help in performance analysis and are centrally monitored on SAP Solution Manager. These are displayed as reports where you can extract information on failure of delivering a request, time taken for a request to reach a component and so on.

On the client side, the client framework enables an application developer to switch on the trace for messages. The client traces the request at predefined points and all these transactions/ requests are recorded in a Business Transaction XML. Additionally, the client maintains a unique identifier in the HTTP header called the SAP Passport that is used to correlate traces across various components. This Business Transaction XML can later be uploaded to the SAP Solution Manager which is a central location to correlate all logging information.

The end to end tracing APIs are available in the `SMPClientConnection` class. For more information on the significance of different trace levels, see *REST SDK API Usage > Rest Client API Reference for Android > PropertyID class*.

Syntax

- Use the `setTraceLevel` API to set the trace level in the SAP-PASSPORT and BTX. The trace level can be set to TrcLvl_HIGH, TrcLvl_MEDIUM, TrcLvl_LOW and TrcLvl_NONE.
  ```
  public void setTraceLevel(int level);
  ```
- Use the `startTrace` API to start end to end tracing on the client.
  ```
  public void startTrace();
  ```
- Use the `stopTrace` API to stop end to end tracing on the client.
  ```
  public void stopTrace() throws SMPException;
  ```
- Use the `uploadTrace` API to generate BTX on the client, and upload the BTX to Solution Manager via the SAP Mobile Platform server.
  ```
  public void uploadTrace() throws SMPException;
  ```

Examples

- **Examples –** Set trace level
  ```
  clientConnection.setTraceLevel(PropertyId.TrcLvl_MEDIUM);
  ```
  Start trace level
  ```
  clientConnection.startTrace();
  ```
  Stop trace level
  ```
  Try{
  clientConnection.stopTrace();
  }catch(SMPException e){
  e.printStrackTrace();
  }
  ```
  Upload trace level
  ```
  Try{
  clientConnection.uploadTrace();
  }catch(SMPException e){
  e.printStackTrace();
  }
  ```

Deleting a User

Delete a registered user.

The `deleteUser` method is used to delete a registered (onboarded) user.

Syntax

```
public void deleteUser()
```

Examples

- **Example for deleting a user**

  ```
  userManager.deleteUser();
  ```

REST SDK API Usage

Rest Client API Reference for Android

Use the Rest Client API reference as the primary reference for all API listings and error code information.

Refer to the Rest Client API reference for each available package.

AppSettings class

Consists of methods used to retrieve setting details required by the application.

Syntax

`public class  AppSettings`

getConfigPropertyMap() method

Syntax

`String getConfigPropertyMap ()` throws `SMPException`

setConfigProperty(HashMap< String, String >) method

Set the value(s) of properties sent as hash map of key and value pairs.

Syntax

`void setConfigProperty ( HashMap< String, String >` *list* `)` throws `SMPException`

Exceptions

- **SMPException class –**

Examples

- **Example**

  ```
  try{
  AppSettings appSettings = new AppSettings(clientConnection);
  HashMap<String,String> property = new HashMap<String,String>();
  property.put("d:AndroidGcmRegistrationId", "1234565");
  appSettings.setConfigurationProperty(property);
  }
  catch(SMPException e){
  ```

```
e.printStackTrace();
}
```

ClientConnection class

Consists of methods used for client-server connection.

Syntax

`public class  ClientConnection`

ClientConnection(Context, String, String, String, SDMRequestManager) constructor

Initialize the client connection with application context, application name, domain,security configuration and request manager.

Syntax

`ClientConnection ( Context` *context* `,  String` *appID* `,  String` *Domain* `,`
`String` *securityConfig* `,  SDMRequestManager` *requestManager* `)`

Parameters

- **context –** Application context.
- **appID –** Application ID.
- **Domain –** Domain name.
- **securityConfig –** Security configuration of the application.
- **requestManager –** Instance of SDMRequestManager.

Examples

- **Example**

```
public static SDMLogger logger;
public static SDMPreferences pref;
public static SDMConnectivityParameters param;
public static SDMRequestManager reqMan;
logger = new SDMLogger();
pref = new SDMPreferences(getApplicationContext, logger);
pref.setBooleanPreference(ISDMPreferences.SDM_PERSISTENCE_SECUREM
ODE, false);
pref.setIntPreference(ISDMPreferences.SDM_CONNECTIVITY_HTTPS_PORT
, 443);
pref.setIntPreference(ISDMPreferences.SDM_CONNECTIVITY_CONNTIMEOU
T, 70000);
pref.setIntPreference(ISDMPreferences.SDM_CONNECTIVITY_SCONNTIMEO
UT, 70000);
param = new SDMConnectivityParameters();
param.setUserName(<username>);
param.setUserPassword(<password>);
reqMan = new SDMRequestManager(logger, pref, param, 1);
ClientConnection clientConnection = new
```

```
ClientConnection(getApplicationContext(), <Application id>="">,
<Domain>, <SecurityConfig>, <RequestManager object>="">);
```

Usage

Note: Setting username and password in requestManager as part of param is mandatory.

getApplicationID() method

Return Application ID.

Syntax

`String getApplicationID ()`

Examples

- **Example**

  ```
  clientConnection.getApplicationID();
  ```

getDomain() method

Return Domain.

Syntax

`String getDomain ()`

Examples

- **Example**

  ```
  clientConnection.getDomain();
  ```

getRequestManager() method

Return Request Manager object.

Syntax

`SDMRequestManager getRequestManager ()`

Examples

- **Example**

  ```
  clientConnection.getRequestManager();
  ```

getSecurityConfig() method
Return Security Configuration.

Syntax
`String getSecurityConfig ()`

Examples

- **Example**

```
clientConnection.getSecurityConfig();
```

getServerHost() method
Return Server host.

Syntax
`String getServerHost ()`

Examples

- **Example**

```
clientConnection.getServerHost();
```

setApplicationConnectionID(String) method
Set the application connection ID.

Syntax
`void setApplicationConnectionID ( String` *appConID* `)`

Parameters

- **appConID –** Application connection ID.

Examples

- **Example**

```
clientConnection.setApplicationConnectionID("myAppConID");
```

Usage

If it is set before invoking userManager.register() interface, the user is registered using this application connection ID.

setConnectionProfile(String) method

Set the URL of the server.

Syntax

`void setConnectionProfile ( String` *url* `)`

Parameters

- **url –** URL of the server.

Examples

- **Example**

```
clientConnection.setConnectionProfile("http://<Host>:<port>");
```

setConnectionProfile(boolean, String, String, String, String) method

Set the value of host and port for the connection.

Syntax

`void setConnectionProfile ( boolean` *isHttpRequest* `, String` *host* `, String` *port* `, String` *relayserverURLTemplate* `, String` *farmID* `)`

Parameters

- **isHttpRequest –** - whether HTTP or HTTPS
- **host –** - host of the server
- **port –** - port of the server
- **relayserverURLTemplate –** URL Template of Relay server
- **farmID –** Farm ID of the relay server

Examples

- **Example**

```
clientConnection.setConnectionProfile(true,<Host>,<port>, <Relay
server URL template>, <Relay server Farm Id>);
```

setTraceLevel(int) method

Set the end to end trace level.

Syntax

`void setTraceLevel ( int` *level* `)`

Parameters

- **level –** Trace level defined in PropertyID class. It includes: TrcLvl_HIGH, TrcLvl_MEDIUM, TrcLvl_LOW and TrcLvl_NONE.

Examples

- **Example**

```
clientConnection.setTraceLevel(PropertyID.TrcLvl_HIGH);
```

startTrace() method

Start the end to end tracing.

Syntax

`void startTrace ()`

Examples

- **Example**

```
clientConnection.startTrace();
```

stopTrace() method

Stop the end to end tracing.

Syntax

`void stopTrace ()` throws `SMPException`

Examples

- **Example –** SMPException class clientConnection.stopTrace();

uploadTrace() method

Upload the BTX file to solution manager for end to end tracing.

Syntax

`void uploadTrace ()` throws `SMPException`

Exceptions

- **SMPException class –**

Examples

- **Example**

  ```
  clientConnection.uploadTrace();
  ```

PropertyID class

Consists of properties used for tracing purposes.

Syntax

```
public class  PropertyID
```

TrcLvl_HIGH variable

Constant to indicate High Trace level for End to End Tracing.

Syntax

```
final int TrcLvl_HIGH
```

TrcLvl_LOW variable

Constant to indicate Low Trace level for End to End Tracing.

Syntax

```
final int TrcLvl_LOW
```

TrcLvl_MEDIUM variable

Constant to indicate Medium Trace level for End to End Tracing.

Syntax

```
final int TrcLvl_MEDIUM
```

TrcLvl_NONE variable

Constant to indicate None Trace level for End to End Tracing.

Syntax

```
final int TrcLvl_NONE
```

SMPException class

Consists of exceptions thrown by the SMP REST client library.

Syntax

```
public class  SMPException
```

SMPException(int) constructor
Constructs a SMPException with specified error code.

Syntax
`SMPException ( int` *errorCode* `)`

Parameters

- **errorCode –**

SMPException(String) constructor
Constructs a SMPException with specified error message.

Syntax
`SMPException ( String` *message* `)`

Parameters

- **message –** - error message

SMPException(int, String) constructor
Constructs a SMPException with specified error code and error message.

Syntax
`SMPException ( int` *errorCode* `,  String` *message* `)`

Parameters

- **errorCode –**
- **message –** - error message

getErrorCode() method
Returns the error code of this SMPException.

Syntax
```
int getErrorCode ()
```

toString() method
Returns the complete error description of this SMPException.

Syntax
```
String toString ()
```

ANY_INPUT_FIELD_NULL variable
Any of the input fields passed is null.

Syntax
`final int ANY_INPUT_FIELD_NULL`

APPLICATION_ID_NULL variable
Application is not initialized.

Syntax
`final int APPLICATION_ID_NULL`

APPLICATION_USER_ALREADY_REGISTERED variable
Attempt to register an application user which is already registered.

Syntax
`final int APPLICATION_USER_ALREADY_REGISTERED`

APPLICATION_USER_NOT_REGISTERED variable
Attempt to delete the user which has not been registered.

Syntax
`final int APPLICATION_USER_NOT_REGISTERED`

AUTHENTICATION_ERROR variable
Error while authenticating user.

Syntax
`final int AUTHENTICATION_ERROR`

BTX_UPLOAD_ERROR variable
E2E BTX upload error.

Syntax
`final int BTX_UPLOAD_ERROR`

CAPTCHA_LISTENER_NULL variable
Captcha listener is null.

Syntax
`final int CAPTCHA_LISTENER_NULL`

CLASS_INITIALIZATION_FAILED variable
Class initialization failed.

Syntax
```
final int CLASS_INITIALIZATION_FAILED
```

E2E_NOT_STARTED variable
E2E Not started.

Syntax
```
final int E2E_NOT_STARTED
```

EMPTY_RESPONSE_FROM_SERVER variable
Empty response string received from server.

Syntax
```
final int EMPTY_RESPONSE_FROM_SERVER
```

HTTP_ERROR variable
HTTP error.

Syntax
```
final int HTTP_ERROR
```

INVALID_MODULE_ID variable
Invalid module handle ID.

Syntax
```
final int INVALID_MODULE_ID
```

JSON_PARSING_FAILED variable
Internal error, JSON parsing has failed.

Syntax
```
final int JSON_PARSING_FAILED
```

NETWORK_ERROR variable
Network error: Obtained when the URL is unreachable.

Syntax
```
final int NETWORK_ERROR
```

Remarks
Check AndroidManifest.xml if it has proper permissions.

PARSE_ERROR variable
Parse error.

Syntax
`final int PARSE_ERROR`

REGISTRATION_FAILED_UNKNOWN_ERROR variable
User registration timed out.

Syntax
`final int REGISTRATION_FAILED_UNKNOWN_ERROR`

REGISTRATION_LISTENER_NULL variable
Asynchronous user registration listener not registered with UserManager.

Syntax
`final int REGISTRATION_LISTENER_NULL`

SINGLETON_INITIALIZATION_FAILED variable
singleton Initialization failed

Syntax
`final int SINGLETON_INITIALIZATION_FAILED`

SUP_INTERNAL_PARSING_ERROR variable
SUP Internal Parsing Error.

Syntax
`final int SUP_INTERNAL_PARSING_ERROR`

UPDATE_SETTINGS_CONTAINS_READONLY_FIELDS variable
Update Settings consists of one or more read only fields.

Syntax
`final int UPDATE_SETTINGS_CONTAINS_READONLY_FIELDS`

UserManager class

Consists of methods used to register or uregister a user.

Syntax

`public class UserManager`

Remarks

Additionally, there are APIs used to provision settings on the client. An application developer can later register a user using these settings.

UserManager(ClientConnection) constructor

Initialize the UserManager.

Syntax

`UserManager ( ClientConnection` *clientCon* `)`

Parameters

- **clientCon –** ClientConnection

Examples

- **Example**

```
UserManager userManager = new UserManager(clientConnection)
```

deleteUser() method

Delete the current user from the SMP server.

Syntax

`void deleteUser ()` throws `SMPException`

Exceptions

- **SMPException class –**

Examples

- **Example**

```
UserManager userManager = new UserManager(clientConnection);
userManager.deleteUser();
```

getApplicationConnectionId() method

Return the application connection ID obtained from the server after successful registration.

Syntax

`String getApplicationConnectionId ()` throws `SMPException`

Examples

- **Example**

```
UserManager userManager = new UserManager(clientConnection);
userManager.getApplicationConnectionId();
```

isUserRegistered() method

Return whether the user is registered or not.

Syntax

`boolean isUserRegistered ()`

Examples

- **Example**

```
UserManager userManager = new UserManager(clientConnection);
if(false == userManager.isUserRegistered()){
userManager.registerUser(true);
}
```

registerUser(boolean) method

Register the user with SAP Mobile Server.

Syntax

`boolean registerUser ( boolean` *isSynchronous* `)` throws `SMPException`

Parameters

- **isSynchronous –** Specifies whether it is synchronous or asynchronous registration.

Exceptions

- **SMPException class –**

Examples

- **Example**

```
Try{
UserManager userManager = new UserManager(clientConnection);
userManager.registerUser(true);        //for synchronous
registration
}
Catch(SMPException e){
}
Try{
UserManager userManager = new UserManager(clientConnection);
userManager.setUserRegistrationListener(this);
userManager.registerUser(false);     //for Asynchronous
registration
}
Catch(SMPException e){
}
public void onAsyncRegistrationResult(State
registrationState,ClientConnection conn, int errCode, String
errMsg) {
TODO Auto-generated method stub
if(registrationState == State.SUCCESS)
{
Log.i("RegistrationState Success",conn.getServerHost()
+conn.getApplicationID()+conn.getDomain()
+conn.getRequestManager().toString());}
if(registrationState == State.FAILURE)
{
Log.i("Registration State Failure",errCode+errMsg
+conn.getServerHost());
}
}
```

Usage

It requires to send username and password as a part of request manager during client connection initialization. If asynchronous, the user has to implement and register 'IODPUserRegistrationListener' to retrieve a callback.

registerUser(String, boolean) method
Register the user with the SAP Mobile Server.

Syntax

`boolean registerUser ( String` *captcha* `, boolean` *isSynchronous* `)` throws `SMPException`

Parameters

- **isSynchronous –** Specifies whether it is synchronous or asynchronous registration.
- **captcha –** The CAPTCHA text obtained in the first registration call.

Exceptions

- **SMPException class –**

Examples

- **Example**

```
Try{
UserManager userManager = new UserManager(clientConnection);
UserManager.registerUser(<captcha text>,true);
}
Catch(SMPException e){
}
```

Usage

It requires to send username and password as a part of request manager during client connection initialization. CAPTCHA text should be passed to the user during the first registration by the CAPTCHA challenge listener callback. If asynchronous, the user has to implement and register ‘IODPUserRegistrationListener’ to retrieve a callback.

setCaptchaChallengeListener(ISMPCaptchaChallengeListener) method

Set the CAPTCHA challenge listener in case of CAPTCHA enabled by the SAP Mobile Server.

Syntax

`void setCaptchaChallengeListener`
`( ISMPCaptchaChallengeListener` *captchaListener* `)`

Parameters

- **captchaListener –** SMP CAPTCHA challenge listener to be registered to retrieve a callback.

Examples

- **Example**

```
UserManager userManager = new UserManager(clientConnection);
userManager.setCaptchaChallengeListener(<ISMPCaptchaChallengeList
ener Object>);
```

Usage

User has to implement the ‘IODPCaptchaChallengeListener’ and set it using the above listener to get the captcha callback. The callback method ‘validateCaptcha’ has to return the CAPTCHA text for registering the user. Alternately, it can also return ‘null’ in which case, the captcha text has to be stored by the application and a subsequent registerUser(captchaText, isSynchronous) has to be explicitly called.

setUserRegistrationListener(ISMPUserRegistrationListener) method

Set the registration listener for asynchronous registration.

Syntax

```
void setUserRegistrationListener
( ISMPUserRegistrationListener  registrationListener )
```

Parameters

- **registrationListener –** SMP user registration listener.

Examples

- **Example**

  ```
  UserManager userManager = new UserManager(clientConnection);
  userManager.setUserRegistrationListener(<ISMPUserRegistrationList
  ener Object>);
  ```

Usage

User has to implement the 'ISMPUserRegistrationListener' and set it using the above listener to get the registration callback.

SMPClientListeners interface

Consists of listeners used by SMP REST client library.

Syntax

```
public interface  SMPClientListeners
```

ISMPCaptchaChallengeListener interface

Interface to get the Captcha Challenge.

Syntax

```
public interface  ISMPCaptchaChallengeListener
```

validateCaptcha(String) method

Call back method to return the CAPTCHA text for registering the user.

Syntax

```
String validateCaptcha ( String image )
```

Parameters

- **image –** Base64 encoded Captcha image.

Examples

- **Example**

```
public String validateCaptcha(String image){
//Show the image to the user.
//Get the Captcha text
return <captchaText>;
}
```

Usage

Alternately, it can also return ‘null’ in which case, the captcha text has to be stored by the application and a subsequent registerUser(captchaText, isSynchronous) has to be explicitly called on

UserManager

.

ISMPUserRegistrationListener interface

Interface to get the result of Asynchronous Registration call.

Syntax

`public interface ISMPUserRegistrationListener`

onAsyncRegistrationResult(State , ClientConnection , int, String) method

Call back method upon Asynchronous Registration call.

Syntax

`void onAsyncRegistrationResult ( State` *registrationState* `, ClientConnection` *clientConnection* `, int` *errorCode* `, String` *errorMsg* `)`

Parameters

- **registrationState –** represents the status of the Asynchronous Registration call.
- **clientConnection –** ClientConnection object to uniquely identify the Asynchronous Registration call.
- **errorCode –** Error Code in case of registrationState being 'FAILURE'.
- **errorMsg –** Error Message in case of registrationState being 'FAILURE'.

Examples

- **Example**

```
Try{
UserManager userManager = new UserManager(clientConnection);
userManager.setUserRegistrationListener(this);
userManager.registerUser(false);     //for Asynchronous
registration
}
Catch(SMPException e){
}
public void onAsyncRegistrationResult(State
registrationState,ClientConnection conn, int errCode, String
errMsg) {
TODO Auto-generated method stub
if(registrationState == State.SUCCESS)
{
Log.i("Registration State Success",conn.getServerHost()
+conn.getApplicationID()+conn.getDomain()
+conn.getRequestManager().toString());}
if(registrationState == State.FAILURE)
{
Log.i("Registration State Failure",errCode+errMsg
+conn.getServerHost());
}
}
```

State() enumeration

Enum to represent the status of Asynchronous Registration call.

Enum Constant Summary

- **SUCCESS –** Successful registration state.
- **FAILURE –** Failure registration state.

ISMPCaptchaChallengeListener interface

Interface to get the Captcha Challenge.

Syntax

`public interface  ISMPCaptchaChallengeListener`

validateCaptcha(String) method

Call back method to return the CAPTCHA text for registering the user.

Syntax

`String validateCaptcha ( String` *image* `)`

Parameters

- **image –** Base64 encoded Captcha image.

Examples

- **Example**

```
public String validateCaptcha(String image){
//Show the image to the user.
//Get the Captcha text
return <captchaText>;
}
```

Usage

Alternately, it can also return ‘null’ in which case, the captcha text has to be stored by the application and a subsequent registerUser(captchaText, isSynchronous) has to be explicitly called on

UserManager

.

ISMPUserRegistrationListener interface

Interface to get the result of Asynchronous Registration call.

Syntax

`public interface ISMPUserRegistrationListener`

onAsyncRegistrationResult(State , ClientConnection , int, String) method

Call back method upon Asynchronous Registration call.

Syntax

`void onAsyncRegistrationResult ( State` *registrationState* `, ClientConnection` *clientConnection* `, int` *errorCode* `, String` *errorMsg* `)`

Parameters

- **registrationState –** represents the status of the Asynchronous Registration call.
- **clientConnection –** ClientConnection object to uniquely identify the Asynchronous Registration call.
- **errorCode –** Error Code in case of registrationState being 'FAILURE'.
- **errorMsg –** Error Message in case of registrationState being 'FAILURE'.

Examples

- **Example**

```
Try{
UserManager userManager = new UserManager(clientConnection);
userManager.setUserRegistrationListener(this);
userManager.registerUser(false);     //for Asynchronous
registration
}
Catch(SMPException e){
}
public void onAsyncRegistrationResult(State
registrationState,ClientConnection conn, int errCode, String
errMsg) {
TODO Auto-generated method stub
if(registrationState == State.SUCCESS)
{
Log.i("Registration State Success",conn.getServerHost()
+conn.getApplicationID()+conn.getDomain()
+conn.getRequestManager().toString());}
if(registrationState == State.FAILURE)
{
Log.i("Registration State Failure",errCode+errMsg
+conn.getServerHost());
}
}
```

State() enumeration
Enum to represent the status of Asynchronous Registration call.

Enum Constant Summary

- **SUCCESS –** Successful registration state.
- **FAILURE –** Failure registration state.

Enabling Mixed Connectivity Using Messaging Channel and HTTP Channel (REST SDK)

Use `SDMConstants` interface in `SDMPreferences` class to set the request-response between messaging channel or REST SDK.

To enable request-response using REST SDK, set the constant `ISDMPreferences.SDM_CONNECTIVITY_HANDLER_CLASS_NAME` to `SDMConstants.SDM_HTTP_HANDLER_CLASS` in the `SDMPreferences` class which is used to initialize the `SDMRequestManager` class.

```
sdmpreferenceObject.setStringPreference(ISDMPreferences.SDM_CONNECT
IVITY_HANDLER_CLASS_NAME, SDMConstants.SDM_HTTP_HANDLER_CLASS);
reqMan = new SDMRequestManager(sdmloggerObject, sdmpreferenceObject,
sdmparameterObject, 1);
```

To enable request-response using messaging channel, either do not set `ISDMPreferences.SDM_CONNECTIVITY_HANDLER_CLASS_NAME`. If the constant is not set, default preference will be used in messaging channel. Otherwise, the constant can be set to `SDMConstants.SDM_IMO_HANDLER_CLASS`.

See also

- *Development Task Flow Using REST SDK (HTTP Channel)* on page 196

Deploying Applications to Devices

This section describes how to deploy customized mobile applications to devices.

1. *Installing Applications on the Device without Using the Google Play*
   Connect the device to your personal computer and install applications without using the Google Play.
2. *Installing Applications using a URL*
   Install applications on an Android device without using the Android market.

See also

- *OData SDK Components and APIs* on page 229

Installing Applications on the Device without Using the Google Play

Connect the device to your personal computer and install applications without using the Google Play.

Prerequisites

- Activate the installation of programs on your device that do not originate from the Google Play. Navigate to **Settings** > **Applications** > **Unknown Sources** to allow installation of these programs.

Task

1. Download the driver software and install this on your personal computer.
   Example: HTC Sync for all HTC Android Phones and HTC Smart Phones, see *http://www.htc.com/uk/help*.
2. Connect your device to the personal computer through a USB cable.
   The driver software uploads the device software and installs it on the device.
3. On the device display screen, make the selection to enable to mount the memory card.
4. Copy the .apk file to the memory card.

5. Disconnect the USB cable from the device.
6. Using the file manager on the device, access the .apk file from the memory card and follow the instructions as displayed.

Installing Applications using a URL

Install applications on an Android device without using the Android market.

Prerequisites

- Activate the installation of programs on your device that do not originate from the Android market. Navigate to **Settings > Applications > Unknown Sources** to allow installation of these programs.
- You must have the URL where the Android package is available as a resource.

Task

1. Enter the URL details in the device browser.
2. Follow the instructions displayed on the browser to install the application.

OData SDK Components and APIs

The Android OData SDK provides a set of features that help application developers build new applications on top of the Android platform. It supports the usage of the OData protocol with SAP additions (OData for SAP) and provides solutions for the most common use-cases an application developer meets with.

Prerequisites for Developing Android Apps

Download the Android Software Development Kit. The recommended development environment is Eclipse IDE (version 3.5 and higher). Also download the Android java plug-in for Eclipse. For more details about Android SDK end Eclipse plug-in installation, see: *http://developer.android.com/sdk/installing.html*

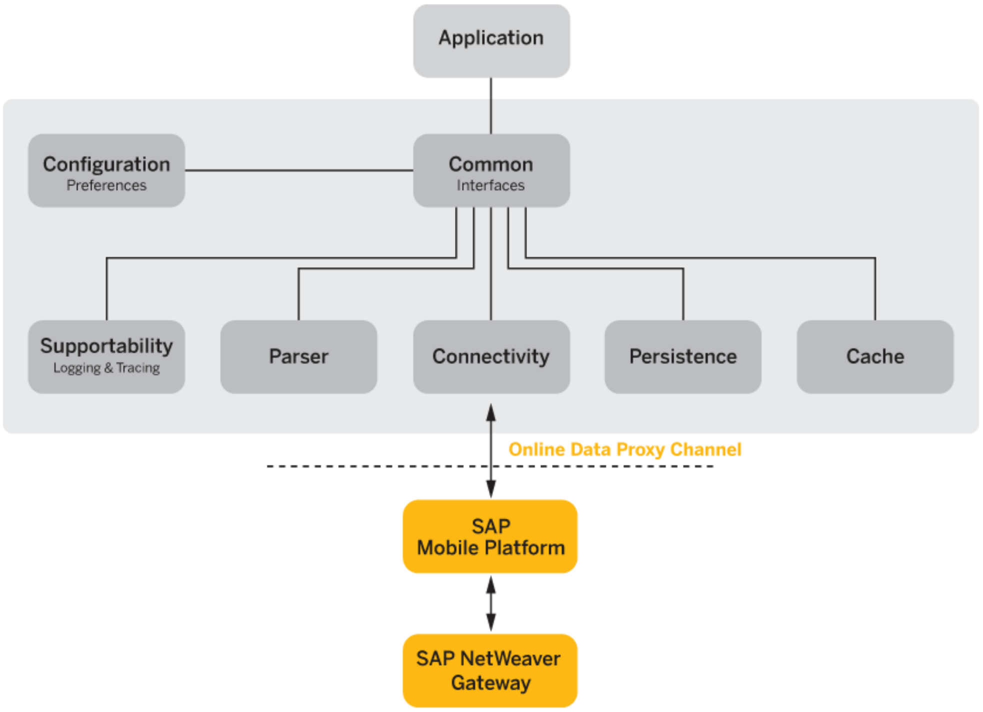
The Android OData SDK also provides emulator support for testing, however, in Android platform, debugging and testing on real devices are more effective. To deploy your application directly to a real device, first install the driver of the device on your computer. For debugging an application on a real device, change the settings of your device to accept non-market applications. (You can change the setting at `Settings > Application > Development`)

Each component of the Android OData SDK can be imported to your project as an external library. The components are built on top of the Android SDK with API level 8.

OData SDK - Android

The full list of APIs and their descriptions are available after the installation of SAP Mobile Platform at the following location within your installation folder:*`SMP_HOME`*`\MobileSDK<Version>\OData\Android\docs`

The following figure shows the main components of the OData SDK on Android.



Each component is implemented as a standalone Java project, so they are available for application developers as separate external libraries (jar files). You also need the SDMCommon component to be able to reuse any other components from the Android OData SDK.

SDMCommon

To build an application on the OData SDK, you must first import the SDMCommon component that contains interfaces and configuration for the components. None of the components have dependency on each other, but all of them depend on the SDMCommon component, and all of them have references to interfaces of other components (held by SDMCommon).

Component Replacements

In your own application, you can replace the implementation behind an interface of an Android OData SDK component. For example, if you want to add a new functionality to SDMCache, but keep everything else unchanged (for example, the way it is persisted by SDMPersistence) you can implement your own solution. The new cache can be either a new implementation, or a descendant of SDMCache, as long as it implements the ISDMCache interface from SDMCommon.

See also

- *Deploying Applications to Devices* on page 228

SDMParser

The SDMParser component is responsible for transforming between the different representations of OData structures, for example, parsing from XMLs to a Java Object or building XMLs from a Java Object.

List of Features

- Parsing OData XML structures to OData Java Objects
- Providing direct access to the most common OData fields and structures in the Java Objects that are the outcome of parsing
- Providing dynamic access to all OData fields and structures in the Java Objects that are the outcome of parsing
- Building OData XML structures from OData Java Objects
- Partial validation of OData XMLs

SDMParser Public APIs

Provides a list of public APIs in SDMParser library.

SDMParser Interfaces

```
ISDMParser
ISDMODataServiceDocument parseSDMODataServiceDocumentXML(String
serviceDocumentXML)
ISDMODataServiceDocument
parseSDMODataServiceDocumentXML(InputStream stream)

ISDMODataSchema parseSDMODataSchemaXML(String schemaXML,
ISDMODataServiceDocument serviceDocument)
ISDMODataSchema parseSDMODataSchemaXML(InputStream stream,
ISDMODataServiceDocument serviceDocument)

List<ISDMODataEntry> parseSDMODataEntriesXML(String entriesXML,
String collectionId, ISDMODataSchema schema)
List<ISDMODataEntry> parseSDMODataEntriesXML(InputStream stream,
String collectionId, ISDMODataSchema schema)
```

```
ISDMODataOpenSearchDescription
parseSDMODataOpenSearchDescriptionXML(String
openSearchDescriptionXML, String collectionId,
ISDMODataServiceDocument serviceDocument)
ISDMODataOpenSearchDescription
parseSDMODataOpenSearchDescriptionXML(InputStream stream, String
collectionId, ISDMODataServiceDocument serviceDocument)

ISDMODataError parseSDMODataErrorXML(String errorXML)
ISDMODataError parseSDMODataErrorXML(InputStream stream)

List<ISDMODataEntry> parseFunctionImportResultXML(String xml,
ISDMODataFunctionImport functionImport, ISDMODataSchema schema)
List<ISDMODataEntry> parseFunctionImportResultXML(InputStream
stream, ISDMODataFunctionImport functionImport, ISDMODataSchema
schema)

String buildSDMODataEntryXML(ISDMODataEntry entry)
String buildSDMODataDocumentXML(ISDMParserDocument document)

ISDMParserDocument parseXML(String xml)
ISDMParserDocument parseXML(InputStream stream)
```

SDM Parser APIs

```
void enableParserPerformanceLog(boolean perflog,
android.content.Context applicationContext);
    void readNameMap();
    void writeNameMap(int starpos);
    public abstract void putComplexPropertyValue(String
propertyValue,String... propertyName) ;
```

Technical Details

The SDMParser component uses javax.xml.parsers.SAXParser as a parser engine, defining its own extension of `org.xml.sax.helpers.DefaultHandler` class as a handler for SAXParser.

The outcome documents of SDMParser are all optimized for persistence using SDMPersistence, implementing the `ISDMPersistable` interface.

To support optimized performance and ensure consistent behavior, SDMParser can persist parsing related data on the device. End users can not delete parser related persisted data, unless they uninstall the whole application. To set the default folder of SDMParser's persistence, change the default value of the appropriate preference: PARSER_DEFAULTFOLDER_PATH (see more at the section about the SDMConfiguration component of the Android OData SDK).

Parsing related data is loaded during the initialization of the SDMParser component. This means that SDMParser must always be initialized before using the SDMParser documents.

As a result of parsing, SDMParser provides Java Object representations of the appropriate OData structures. Each such SDMOData Java Object is a representation of the appropriate

Data XML and provides dynamic access to all of its elements and attributes. Besides the full access with the dynamic method, OData Java Objects provide interfaces for a more convenient access of data used in the most common scenarios.

ETag Support

An entity tag is one of the several mechanisms that HTTP provides for cache validation and which allows a client to make conditional requests. An ETag is an identifier assigned by a Web Server to a specific version of a resource found at a URL. If the resource content at the URL changes, a new and different ETag is assigned.

Examples

Provides example snippets to understand the use of SDMParser APIs.

Example of parsing a feed or single entry

```
   try {
    //Parsing a feed or a single entry.
    //Assuming that schema and service document already parsed
    //and collection selected
    List<ISDMODataEntry> entries =
       parser.parseODataEntriesXML(responseXML,collectionId,schema);
    //Assuming there is at least one entry in the feed.
    ISDMODataEntry entry = entries.get(0);
    //Retrieving the valid property meta data from the given
SDMOdataSchema.
    List<ISDMODataProperty> properties = entry.getPropertiesData();
    //Assuming there is at least one property for the entry.
    ISDMODataProperty property = properties.get(0);
    boolean visibleInList = property.getAttribute("visible-in-
list");
    String value;
    if (visibleInList) {
        value = property.getValue();
    } else {
        value = "invisible";
    }
   } catch(SDMParserException e) {}
```

ETag Support

```
private SDMODataEntry entryDoc;  //entryDoc corresponds to the XML
document with entries
String eTag = entryDoc.getEtag();  //Passes a null value if the
document does not contain an ETag.
```

SDMCache

The SDMCache component is responsible for storing and accessing OData related objects in the memory of the device.

List of Features

- Storing SDMOData document objects in the memory
- Accessing SDMOData documents in the memory directly by their key
- Searching for SDMODataEntry objects in the memory using tokenized prefix search on their searchable fields
- Searching for SDMODataEntry objects in the memory using one of the following predefined algorithm: full term prefix search, tokenized contain search, full term contain search, tokenized contain-all search and regex search
- Searching for SDMODataEntry objects in the memory using custom search algorithm
- Managing the number of stored SDMOData documents based on the maximum size of the capacity, removing the least recently used SDMOData document first
- Validating the references between the stored SDMOData Service Document, SDMOData Schema and the SDMOData Entries

SDMCache Public APIs

Provides a list of public APIs available in the SDMCache library.

SDMCache Public APIs

```
ISDMCache

    void clear();
    void setSDMODataServiceDocument(ISDMODataServiceDocument 
serviceDocument);
    void setSDMODataSchema(ISDMODataSchema schema);
   void setSDMODataEntry(ISDMODataEntry entry, String collectionId);
    void setSDMODataEntries(List<ISDMODataEntry> entries, String 
collectionId);
    void setSDMODataDocument(ISDMParserDocument document);
    ISDMODataServiceDocument getSDMODataServiceDocument();
    ISDMODataSchema getSDMODataSchema();
    ISDMODataEntry getSDMODataEntry(String key);
    List<ISDMODataEntry> getSDMODataEntries(String collectionId);
    ISDMParserDocument getSDMODataDocument(String key);
    List<ISDMODataEntry> searchSDMODataEntries(String searchTerm, 
String collectionId);
    void removeSDMODataServiceDocument();
    void removeSDMODataSchema();
    void removeSDMODataDocument(String key);
    void removeSDMODataEntries(String collectionId);
    void removeStoredDocuments();
    int getSearchAlgorithm();
    void setSearchAlgorithm(int searchAlgorithm);
    void setEntrySearch(ISDMEntrySearch entrySearch);
```

Technical Details

For capacity management, SDMCache uses an LRU (least recently used) algorithm that ensures that the most recently used entries will not be removed first because of reaching the maximum capacity. Maximum number of capacity can be set using preference with key: ISDMPreferences.SDM_CACHE_CAPACITY. This setting refers to the maximum number of cached entities per Collection.

SDMCache supports several predefined search algorithms optimized for performance. Application developers can also set their own EntrySearch object in order to use a custom search algorithm.

SDMCache is an implementer of the `ISDMPersistable` interface, so it can be persisted with the SDMPersistence component (see more at the section about the SDMPersistence component of Android OData SDK).

SDMCache validates the incoming SDMOData documents by matching their references to each other. A single SDMCache object can store only one set of documents (one Service Document with one related Schema with any number of related Entries).

The SDMOData document created by the SDMParser component automatically sets the required references to the related objects. All these references are automatically maintained during persisting or loading the SDMCache using SDMPersistence.

SDMCache depends on the SDMOData specific interfaces of SDMParser, but does not depend on the real implementation of SDMParser.

SDMPersistence

The Persistence component is responsible for storing application specific objects and raw data in the device's physical storage.

List of Features

- Storing objects and raw data on the physical storage of the device
- Accessing objects and raw data stored on the physical storage of the device
- Encrypting and storing objects and raw data on the physical storage of the device using the secret key provided by the application developer
- Generating initial secret key
- Accessing and decrypting objects and raw data stored encrypted on the physical storage of the device using the secret key provided by the application developer

SDMPersistence Public APIs

Provides a list of public APIs in the SDMPersistence library.

SDMPersistence Public APIs

```
ISDMPersistence
```

```
    void clear();
    void storeObject(String key, ISDMPersistable object)
    <T extends ISDMPersistable> boolean loadObject(String key, T
object)
    void storeRawData(String key, byte[] data)
    byte[] loadRawData(String key)
    void storeDataStream(String key, InputStream stream)
    InputStream loadDataStream(String key)
    boolean loadSDMCache(ISDMCache cache)
    boolean loadSDMCache(String key,ISDMCache cache)
    void storeSDMCache(String key, ISDMCache cache)
    boolean removeData(String key)
    boolean removeSDMCache()
    boolean removeSDMCache(String key)
    boolean isDataPersisted(String key)
    void setEncryptionKey(byte[] secretKey, String
secretKeyAlgorithm) throws SDMPersistenceException
```

SDMPersistable Public APIs

```
ISDMPersistable
    void write(BufferedWriter writer)
    void read(BufferedReader reader)
```

SDMCacheLoadListener Public APIs

```
ISDMCacheLoadListener
    void onSDMODataServiceDocumentLoaded(ISDMCache cache,
ISDMODataServiceDocument serviceDocument)    void
read(BufferedReader reader)
    void onLoadFinished(ISDMCache cache)
    void onSDMODataEntriesLoaded(ISDMCache cache,
List<ISDMODataEntry> entries)
    void onLoadFinished(ISDMCache cache)
```

Technical Details

SDMPersistence preferences:

SDMPersistence can persist data in secure and non-secure mode, based on the value of preference PERSISTENCE_SECUREMODE_BOOLEAN. In secure mode, all data stored by SDMPersistence will be encrypted. Encryption is done using the secret key that is passed to SDMPersistence during initialization or by using the `setEncryptionKey` API. If the Secret Key object is null and the secure mode is turned on for SDMPersistence, SDMPersistenceException will be thrown during runtime. PERSISTENCE_SECUREMODE_BOOLEAN is by default true.

A secret key can be generated with the help of the static API of the `SDMPersistence` class. Use the `SDMPersistence.generateSecretKey (String secretKeyAlgorithm)` API. The API returns a generated secret key with the given algorithm in a byte array format.

Important: if PERSISTENCE_SECUREMODE_BOOLEAN preference is changed during runtime, all previously stored data will be deleted without any notification. It depends on the application whether it asks for confirmation from the user before changing the value of this preference.

SDMPersistence by default stores data in the application's cache folder in the file system on the physical storage of the device. Stored data is not accessible for any other applications, but can be wiped out by the user outside of the application using the device's application settings. The default folder to store persisted data can be changed by changing the value of PERSISTENCE_DEFAULTFOLDER_PATH_STRING preference. If the PERSISTENCE_DEFAULTFOLDER_PATH_STRING preference is changed during runtime, all previously stored data will be automatically moved to the new folder.

SDMPersistence implementation guarantees the proper concurrent file handling as long as there are no other non SDMPersistence objects trying to access the persisted data.

`ISDMPersistable`:

All the objects that are to be persisted with SDMPersistence need to implement the ISDMPersistable interface. All valid implementations of ISDMPersistable must implement the declared read and write methods of the interface and must have a public no-arg constructor. SDMOData objects provided by the Android OData SDK are valid implementations of ISDMPersistable.

SDMConnectivity

The SDMConnectivity layer hides the complexity of network communication and provides easy to use APIs to the applications.

List of Features

- Provides interfaces for request handling
- Handles the requests synchronously and asynchronously
- Handles the requests by multiple number of threads (configurable)
- Handles the data streaming from server to device and device to server. A streaming API is provided to allow the value to be accessed in chunks. See *SDMRequest* and *SDMResponse*, for more information on streaming APIs.

SDMConnectivity Public APIs

Provides a list of public APIs in the SDMConnectivity library.

SDMConnectivity Public APIs

Note: The SAP Mobile Platform APIs and their descriptions are available after the installation of SAP Mobile Platform at the following location within your installation folder: `...\UnwiredPlatform\ClientAPI\apidoc`.

The SDMRequestManager class implements the ISDMRequestManager interface, which provides the following methods:

```
ISDMRequestManager

     void makeRequest(final ISDMRequest aRequest);
     void makeRequest(final ISDMBundleRequest aRequest);
     ConnectivityParameters getConnectivityParameter();
     int getQueueSize();
     Vector getAllRequests();
     Object getRequest();
     void setMainHandlerClassName(final String classname);
     void terminate();
     boolean hasRequests();
 void sendOnSuccess(final ISDMNetListener listener, final
ISDMRequest request, finral HttpResponse response);
```

The number of working threads in the RequestManager class is configurable via the constructor. The number of threads is maximized by the connectivity layer because of performance related issues. If the client initializes the layer with more than the allowed threads, the implementation of the connectivity layer will decrease the thread number to the max allowed number (4).

Methods defined by the ISDMConnectivityParameters interface:

```
SDMConnectivityParameters

    void enableXsrf(Boolean isXsrfEnabled)
    void setUserName(String aUserName);
    String getUserName();
    void setUserPassword(String aPassword);
    String getUserPassword();
    void setBaseUrl(final String url);
    String getBaseUrl();
    String getLanguage();
    void setLanguage(String language);
    void setServerCertificate(Certificate certificate) throws
KeyStoreException;
    final TrustManager[] getTrustManagers();
```

Sending requests with the connectivity layer consists of the following steps:

1. Create the RequestManager class and initialize it with the required parameters.
2. Create the request object. This can be done in the following ways:
   - Implement the ISDMRequest interface.
   - Extend the BaseRequest class, which is the base implementation of the ISDMRequest interface.
   - When the requests' execution order is important, implement the ISDMBundleRequest, add the ISDMRequest instances into it, then pass this bundle to the request manager. Both of them are provided by the connectivity layer.
3. Use the request / request bundle object when making a request to the RequestManager.

SDMBundleRequest is a special set of SDMRequest objects. It provides serial processing of the requests when the SDMRequestManager is in multithreaded mode. Because the single SDMRequest objects are processed by multiple threads, the timing of the responses are not consistent. With SDMBundleRequest, one thread processes the bundled requests, guaranteeing that the responses are arriving in the same order as the requests are added to the bundle.

Technical Details

The tasks of the connectivity library have been divided into three main categories:

- Manage the request queues
- Manage the reading writing to the input/output streams
- Manage the platform specific connection creation

The Connectivity component always performs the requests in asynchronous mode. The application's role is to handle the requests in sync mode. The component is able to perform HTTP and HTTPS requests, which you can use for developing and testing purposes, but the default is SUPRequest. The threads in the connectivity library are responsible for taking the requests from the queue (FIFO - First in first out - algorithm) and performing the requests. The number of working threads in the connection pool can be configured in the connectivity layer. The queue is handled by the `SDMRequestManager`, and the working threads take the requests from this queue. Applications are interacting only with the `SDMRequestManager` class; the other components of the connectivity library are not visible to them. The network component consists of three main parts:

- `SDMRequestManager`: responsible for queuing the requests, managing the threads and keeping the connection with applications
- `AbstractConnectionHandler`: responsible for performing the request
- `ConnectionFactory`: responsible for creating and managing platform dependent connections to the server

An application can have more than one `SDMRequestManager` instances, for example, when connecting to two different servers at the same time. To support this scenario, `SDMRequestManager` handles `ConnectionHandler` as a plugin. This kind of plugin needs to implement the ISDMConnectionHandler and implement a constructor taking three parameters: **SDMRequestManager**, **ISDMLogger** implementation and **ISDMPreferences** implementation.

The class name with package is set by `SDMRequestManager.setMainHandlerClassName(String)`, or in SDMPreferences by the ISDMPreferences.SDM_CONNECTIVITY_HANDLER_CLASS_NAME preference key. The default plugin is `com.sybase.mobile.lib.client.IMOConnectionHandler`, which handles connections through SAP Mobile Platform.

There is built-in support for setting the timeout for the socket connection: the application can use the `SDMPreferences` object to modify the value, using the following keys:

- ISDMPreferences.SDM_CONNECTIVITY_CONNTIMEOUT for connection timeout, and
- ISDMPreferences.SDM_CONNECTIVITY_SCONNTIMEOUT for socket connection timeout.

SDMRequest

An `SDMRequest` object wraps all the information needed by the connectivity library to be able to perform the requests.

The connectivity library interacts with the request object to query the necessary information about the headers, the post data, and so on. The connectivity layer also uses the request object to notify the application about the result of the request by using the `ISDMNetListener` interface. The connectivity component provides an interface called `ISDMRequest` and a base implementation of it, called `SDMBaseRequest`. The applications have to extend this interface when creating new application specific requests.

```
ISDMRequest

    void setRequestUrl(String url);
    String getRequestUrl();
    void setRequestMethod(final int reqType);
    int getRequestMethod();
    byte[] getData();
    Hashtable getHeaders();
    void setData(byte[] data);
    void setHeaders(Map<String,String> headers);
    void setPriority(final int value);
    int getPriority();
    boolean useCookies();
    ISDMNetListener getListener();
    void setListener(ISDMNetListener listener);
    void setStreamEnabled(boolean streamEnabled);
    void SetDataStream(InputStream stream)
```

The `setStreamEnabled` API is used by the application to enable streaming for downloading large amount of data from server to device.

The `SetDataStream` API is used by the application to enable streaming for uploading large amount of data from device to server.

The connectivity layer notifies the client about the result of a request by the ISDMNetListener interface. Usage of this feature is not mandatory, but it is recommended to be able to handle incidental errors. Methods available in the ISDMNetListener interface:

```
ISDMNetListener

    void onSuccess(ISDMRequest  aRequest, ISDMResponse aResponse);
    void onError(ISDMRequest aRequest, ISDMResponse aResponse,
SDMRequestStateElement aRequestStateElement);
```

The role of the `SDMRequestStateElement` object used by the connectivity library is to provide the application with more detail on the occurred error. Methods available in ISDMRequestStateElement interface:

```
ISDMRequestStateElement

    int getHttpStatusCode();
    int getErrorCode();
    Exception getException();
    org.apache.http.HttpResponse getHttpResponse()
    void setHttpStatusCode(final int httpStatus);
    void setErrorCode(final int code);
    void setException(final Exception aException);
    void setHttpResponse(org.apache.http.HttpResponse aResponse)
```

```
Example for successful response received by application from EIS

    public void onSuccess(ISDMRequest  aRequest, SDMHttpResponse
aResponse) {
    System.out.println("Http response status code:" +
aResponse.getStatusCode());
    System.out.println("Cookie string:" +
aResponse.getCookieString());
    byte[] content = aResponse.getContent();
    String response = new String(content);
    System.out.println("Received content:" + response);
    //get the headers
    Hashtable headers = aResponse.getHeaders();
}
```

```
Example for uploading data stream from device to server
while((bytesRecvd = resp.getDataStream(buf,102400))>0)
{
   try {
         b.append(new String(buf).trim());
         buf = new byte[102400];
     }
Catch(){
…
}
}
```

SDMResponse

An `SDMResponse` object wraps all the information needed by the connectivity library to be respond back to the application.

The connectivity component provides an interface called ISDMResponse. Available SDMResponse APIs are:

```
ISDMResponse

    boolean isDataAvailable();
    String getCookie();
    int getCurrentOffset();
    void setCorelationId(String coRelationId);
    String getCorelationId();
    int getDataStream(byte[] buffer, int chunkSize);
    int getDataStream(byte[] buffer, int chunkSize, int offset,
String cookie);
```

The `getDataStream` API is used by the application to retrieve the data stream. This API returns the length of data buffer, and the application invokes this API until the length of buffer stream is equal to 0.

```
Example for successful streaming response received by application

public void GetOperation()
{
…
    String collectionUrl = "http://vmw3815.wdf.sap.corp:50009/sap/
opu/sdata/iwfnd/RMTSAMPLEFLIGHT/FlightCollection/";
    getCollection.setRequestUrl(collectionUrl);

getCollection.setRequestMethod(SDMBaseRequest.REQUEST_METHOD_GET);
    getCollection.setListener(this);
    getCollection.setStreamEnabled(true);

    reqMan.makeRequest(getCollection);
…
}

@Override
public void onSuccess(ISDMRequest aRequest, ISDMResponse aResponse)
{
    // TODO Auto-generated method stub
    HttpEntity responseEntity = aResponse.getEntity();
    SDMResponse resp = (SDMResponse)aResponse;
    String reponse = "";
if(aRequest.equals(getCollection))
{
    StringBuffer b = new StringBuffer();
    int bytesRecvd = 0;
    byte[] buf = new byte[102400];


while((bytesRecvd = resp.getDataStream(buf,102400))>0)
{

    try {
        b.append(new String(buf).trim());
        buf = new byte[102400];
 }
    catch (IOException e) {
        e.printStackTrace();
        }
    }

Log.i("NewFlight", "Final streamed data: "+ b.toString());
```

```
    response = b.toString();

    ISDMODataEntry dEntry;
    try {
dEntry = parser.parseSDMODataEntriesXML(response,
"FlightCollection", cache.getSDMODataSchema()).get(0);
        }
catch (IllegalArgumentException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
catch (SDMParserException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

SDMBundleRequest

```
ISDMBundleRequest

    void addRequest(ISDMRequest request);
    void removeRequest(ISDMRequest request);
    void removeRequest(ISDMRequest request);
    void clearRequests();
```

ETag Support

The SDMConnectivity library provides APIs that allow ETags to be forwarded as request headers.

HTTP header fields are components of the message header for requests and responses. These fields define the the operating parameters of an HTTP transaction. The following header fields are applicable for ETag support.

Table 5. ETag Header Fields

Header Fields	Description
If-Match	A client that has one or more ETags obtained from a resource, can verify that one of these ETags is current.
If-None-Match	A client that has one or more ETags obtained from a resource can verify that none of these ETags is current.
If-Range	A client that has an ETag that matches the current ETag of a resource, can request for the specified sub-range of the resource. This means that the client has a partial copy of an entity in its cache and can request to get the entire entity.

Note: If the backend system corresponds to Gateway, only the If-Match header field is supported.

Using ETags for HTTP Request Methods

ETags are forwarded as headers in HTTP request methods. Here are some examples for ETags with some of the commonly used request methods.

- HTTP GET

  This method requests a representation of the specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the value of the specified resource on the server is returned. If it fails, the status code returned is 412 (Precondition Failed)
- HTTP PUT

  This method uploads the representation of the specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the specified resource is uploaded at the server. If it fails, the status code returned is 412 (Precondition Failed).
- HTTP DELETE

  This method deletes a specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the resource is deleted. If it fails, the status code returned is 412 (Precondition Failed).

Examples

Provides example snippets to understand the use of SDMConnectivity APIs.

Creating and adding requests

```
//create and fill parameters for Connectivity library
SDMConnectivityParameters params = new SDMConnectivityParameters();
params.setUserName("test");
params.setUserPassword("testpwd");
mLogger = (ISDMLogger) new SDMLogger();
mPreferences = new SDMPreferences(getApplicationContext(), mLogger);
//create the RequestManager
mRequestManager = new SDMRequestManager(mLogger, mPreferences,
params, 2);
ISDMRequest testRequest = new SDMBaseRequest();
testRequest.setRequestUrl("http://test.de:8080/testpath");
testRequest.setRequestMethod(ISDMRequest.REQUEST_METHOD_GET);
testRequest.setPriority(ISDMRequest.PRIORITY_NORMAL);
//add the request to the connectivity layer
mRequestManager.makeRequest(testRequest);
```

SDMConfiguration

Each low level API has its own defaults/constants set in the SDMCommon library. Default values of preferences can be found in the SDMConstants class.

List of Features

- Providing modifiable preferences for SDMComponent libraries
- Persisting modified values of preferences of SDMComponent libraries
- Validating preferences values of SDMComponent libraries
- Providing API for resetting the preferences of SDMComponent libraries to their default values
- Providing API for creating and handling custom preferences
- Persisting the values of custom preferences
- Notifying subscribed listeners in case of any change in preferences

SDMConfiguration Public APIs

Provides a list of public APIs in the SDMConfiguration library.

SDMConfiguration Public APIs

```
ISDMPreferences

    public void setIntPreference(String key, int value)
    public void setLongPreference(String key, long value)
    public void setFloatPreference(String key, float value)
    public void setBooleanPreference(String key, boolean value)
    public void setStringPreference(String key, String value)

    public void resetPreference(String key)
    public boolean containsPreference(String key)

    public Float getFloatPreference(String key)
    public Integer getIntPreference(String key)
    public Long getLongPreference(String key)
    public Boolean getBooleanPreference(String key)
    public String getStringPreference(String key)

    void registerPreferenceChangeListener(String
key,ISDMPreferenceChangeListener     changeListener)

    void unRegisterPreferenceChangeListener(String
key,ISDMPreferenceChangeListener     changeListener);

    public void removePreference(String key) throws
SDMPreferencesException;
```

```
ISDMPreferenceChangeListener

void onPreferenceChanged(String key,Object value)
```

Technical Details

Android offers an optimized storage for preferences called SharedPreferences (even with automatic Preference screen generation from XML). Modified and custom preferences will be automatically persisted into the default SharedPreferences of the application. Preferences must not contain any secure information. For this purpose, use SDMPersistence in secure mode or the Data Vault from SAP Mobile Platform.

SDMComponents preferences can be reset to their default values using the `resetPreference()` method or by removing them from SharedPreferences.

Any changes to the SDMComponents preferences will be automatically validated regardless whether they are modified by using SDMPreferences or by using the default API of SharedPreferences. If you change the value of a preference to an invalid value while using the SharedPreferences API of the OS, the invalid value will automatically be removed at runtime and the preference will be set to its default value without any notification. Application developers are encouraged to use the API of ISDMPreferences so they will be notified about invalid values.

You can register a preference change listener for each preference in SDMPreferences (including custom preferences) so that you will be notified if the value of a given preference has changed.

Preference change listener notification and preference validation can only be done after the initialization of the appropriate component. It is not recommended to change the values of SDK related preferences outside runtime. For example, if you change the root folder of persisted data at runtime, the SDMPersistence component will automatically move all the persisted data. However, changing this value before the initialization of the SDMPersistence component can result in the loss of persisted data.

The OData SDK provides reusable custom Preference classes as an extension of standard Preference classes provided by the OS for handling Long, Float, and Integer preferences. These custom classes can be reused in preference XMLs or in the custom preferences screen of the application. Custom preference classes can be found in the SDMCommon Component package of the OData SDK.

Supportability

The OData SDK provides a set of features and concepts for the supportability of the applications built on top of the SDK.

Exceptions

Every component of the Library has its own root exception, named as <SDM Library component name>Exception. For instance, in Connectivity, the root exception is SDMConnectivityException. All component-specific exceptions are extending the component’s root exception. Besides root exceptions, SDM components can also throw general exceptions, such as IllegalArgumentException or IllegalStateException.

SDMLogger

The library supports logging via its ISDMLogger interface and provides SDMLogger as an implementation of this interface.

List of Features

- Provides a common interface for handling log messages across the library
- Extends Android's standard logging facility, while keeps method signatures compatible
- Provides facility to store log data
- Provides filterable log retrieval by severity, tag, timestamp (from-to), process id and by correlation id

Technical Details

The interface is similar to Android’s standard logging facility (android.util.Log). Logging does not support security and handling sensitive data. It is the responsibility of the applications to handle these requirements. Logging supports retrieving the log data for persistence or other purposes. `SDMLogger` also implements the ISDMPersistable interface to make the log data persistable. A log header can be set by the application including the following fields:

- Operating System version
- App name
- App Version
- 3rd Party product versions (for example, SQLlite)
- Hardware version
- User
- Timezone
- Language
- SAP Mobile Platform/SAP NetWeaver Gateway URL

The SDMConnectivity sets the User, Language and SAP Mobile Platform/SAP NetWeaver Gateway URL fields. SDMLogger stores log entries timestamped, in milliseconds granularity of the time the log method called by the application/library component. It can also clean out log messages below a certain level, or clean out the log completely. A preliminary log rotation support is built in. At every log method call, a check runs and verifies whether the number of messages reaches 10000. If the number of messages is greater or equal to this threshold, a low priority background thread is started to clean out the oldest 200 log entries.

SDMLogger provides line-level location logging with the full class name of the logging class. Location detection is done by call stack evaluation. Therefore, SDMLogger provides location parameter setting for the logging class, where the class can set the location instead of using the detection facility. Log messages are stored only above the predefined logging level, which defaults to ERROR log level.

Log priority constants:

- PERFORMANCE = 1
- VERBOSE = 2
- DEBUG = 3
- INFO = 4
- WARN = 5
- ERROR = 6
- ASSERT = 7
- FATAL = 8

Log Methods

```
public void log(final int level, final String tag, final String msg,
            final Throwable tr, final String location)
```

Parameters:

```
level  the log level
msg  The message you would like logged.
tr  An exception to log
location  The line-level location of the log source (full class name
of the class)

void  d(String tag, String msg)
void d(String tag, String msg, String location)
void  d(String tag, String msg, Throwable tr)
Void  d(String tag, String msg, Throwable tr, String location)
Sends a DEBUG log message and logs the exception.

void  e(String tag, String msg)
void e(String tag, String msg, String location)
void  e(String tag, String msg, Throwable tr)
void  e(String tag, String msg, Throwable tr, String location)
Sends an ERROR log message and logs the exception.

void  i(String tag, String msg)
void i(String tag, String msg, String location)
void  i(String tag, String msg, Throwable tr)
void  i(String tag, String msg, Throwable tr, String location)
Sends an INFO log message and logs the exception.

void p(String tag, String msg)
void p(String tag, String msg, String location)


void v(String tag, String msg)
void v(String tag, String msg, String location)
void v(String tag, String msg, Throwable tr)
void v(String tag, String msg, Throwable tr, String location)
Sends a VERBOSE log message and logs the exception.

void w(String tag, Throwable tr)
void w(String tag, String msg)
void w(String tag, String msg, String location)
void w(String tag, String msg, Throwable tr)
void w(String tag, String msg, Throwable tr, String location)
void w(String tag, Throwable tr, String location)

Sends a WARN log message and logs the exception.

void wtf (String tag, Throwable tr)
void wtf (String tag, String msg)
void wtf (String tag, String msg, Throwable tr)
void wtf(String tag, Throwable tr, String location)
void wtf (String tag, String msg, Throwable tr, String location)
What a Terrible Failure: Reports a condition that should never
happen. The error will always be logged at level ASSERT.
```

SDMLogger (ISDMLogger implementation that the Library provides) also has the following functionality:

```
public void cleanUp(final int threshold)
Deletes all log entries weaker than the 'threshold' priority.

public void terminate()
Completely clears the collected log data.

void d(String tag, String msg, String location)
void e(String tag, String msg, String location)
void i(String tag, String msg, String location)
void w(String tag, String msg, String location)

public Vector<LogEntry> getLogElements(final int threshold)
This method returns the log data, including all log data with level
'threshold' or above.

public boolean logsToAndroid()
public void logToAndroid(final boolean doIt)
These methods get and set the property which controls sending the log
output to the Android logging facility.

boolean canLog()
int getLogLevel()

void p(String tag, String msg)
void p(String tag, String msg, String location)
void setHeaderData(String osVersion, String appName, String
appVersion, String thirdParty, String hwVersion, String user, String
timezone, String language, String baseUrl)
void setLogLevel(int level)

public boolean logsFullLocation()
public void logFullLocation(boolean logFullLocation)
These methods get and set the property which if full location should
be logged automatically based on the current stack trace.
public synchronized String toString()
Returns all log data – including the header – as String.
```

Sample:

```
Operating System version: 11
Application name: MyApp
Application version: 1.0.0
3rd-party products: -
Hardware version: Galaxy Tab
User name: DEMO
Timezone: CET-DST
Language: en
Base URL: http://www.sap.com/gateway/or/whatever

2011-06-28 14:30:23.368 WARN    SDMPreferences
com.sap.mobile.lib.sdmconfiguration.SDMPreferences.getPreference(SD
MPreferences.java:284)    Deprecated method 'getPreference' has been
called.
2011-06-28 14:30:23.468 INFO    SDMPreferences
com.sap.mobile.lib.sdmconfiguration.SDMPreferences.setStringPrefere
nce(SDMPreferences.java:244)    Preference
'SAP_APPLICATIONID_HEADER_VALUE' (String) has been changed to MyApp.
1.0.0.0
public Vector<LogEntry> getLogElementsByTag(final String aTag)
public Vector<LogEntry> getLogElementsByTimeStamp(final long start,
final long end)
public Vector<LogEntry> getLogElementsByPID(final long PID)
public Vector<LogEntry> getLogElementsByCorrelationId(final String
correlationId)
```

These methods return with a Vector of filtered log entries, filtered by TAG, timestamp (interval), process id and correlation id, respectively.

SAP Passport

For the Single Activity Trace an SAP® Passport has to be issued by the connectivity layer of the library.

The SAP Passport is transported as an HTTP header in the request. The server handles the SAP Passport to generate end-to-end Trace. The OData SDK is using JSDR SAP Passport sources integrated in the library at source level. It can be turned on or off with `ISDMPreferences.SAPPASSPORT_ENABLED` preference key. By default it is turned off.

```
Example
mPreferences = new SDMPreferences(getApplicationContext(), mLogger);
mPreferences.setStringPreference(ISDMPreferences.SAPPASSPORT_ENABLE
D,"true");
```

ODP SDK API Usage

The ODP SDK consists of APIs used to customize applications to send and receive data using Online Data Proxy.

For a comprehensive list of API references, extract the contents from the following zip files:

- *SMP_HOME*\MobileSDK<Version>\OData\Android\docs \AndroidODPSDK-<version>-docs.zip

- *SMP_HOME*`\MobileSDK<Version>\OData\Android\docs\AndroidODataSDK-<version>-docs.zip`

Afaria APIs

Use the Afaria APIs to provision your SAP Mobile Platform application with configuration data for connecting to the SAP Mobile Server, and certificates.

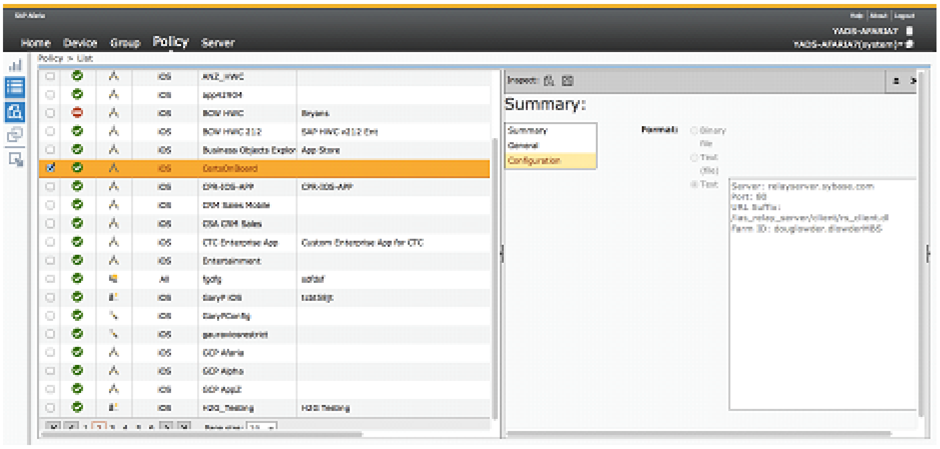
Using Afaria to Provision Configuration Data

You can use Afaria to provision configuration data for a SAP Mobile Platform application, including the SAP Mobile Server server name, port number, and other parameters.

To use these APIs you must provide the application to the device through an Afaria application policy. When setting up such an application policy, the Afaria administration interface provides an option to add configuration data to the policy as text or binary.

The following is an an example of the Afaria administration screen for an application policy that provides an application named "CertsOnBoard" to an enrolled device. The "Configuration" tab shows the configuration data provided to the application.

In this case, the configuration information is added using the administration user interface, but it can also be provided as a text or binary file. The example shows plain text, but you can also provide the information as XML or JSON text for easier parsing by the application.



iOS	ANZ_HWC	
iOS	app42904	
iOS	BOW HWC	Bryans
iOS	BOW HWC 212	SAP HWC v212 Ent
iOS	Business Objects Explor	App Store
iOS	CertsOnBoard	
iOS	CPR-IOS-APP	CPR-IOS-APP
iOS	CRM Sales Mobile	
iOS	CSA CRM Sales	
iOS	CTC Enterprise App	Custom Enterprise App for CTC
iOS	Entertainment	
All	fgdfg	sdfdsf
iOS	GaryP iOS	[illegible]
iOS	GaryPConfig	
iOS	gauravlocrestrict	
iOS	GCP Afaria	
iOS	GCP Alpha	
iOS	GCP App2	
iOS	H2G_Testing	H2G Testing

Server: relayserver.sybase.com
Port: 80
URL Suffix:
/ias_relay_server/client/rs_client.dll
Farm ID: douglowder.dlowderMBS

You can obtain configuration data for your application using Afaria by calling the following API from the `com.sybase.afaria.SeedDataAPI` class (in `AfariaSSL.jar`).

```
String
com.sybase.afaria.SeedDataAPI.retrieveSeedData(SeedDataCredentials
arg0) throws SeedDataAPIException
```

To access this data, the application provides `SeedDataCredentials` to the `retrieveSeedData` API. If the device is correctly enrolled to Afaria, the API returns a string which contains the full path to a file in the application's sandbox with the seed data.

```
SeedDataCredentials sdc = new
SeedDataCredentials(“supadmin”,”xnetqa”,”abc”);
String result = SeedDataAPI.retrieveSeedData(sdc);
resultText.append(“the seed data file: ” + result);
BufferedReader reader = null;
Map<String, String> keyValues = null;
try
{
  reader = new BufferedReader(new FileReader(result));
  String line = null;
  keyValues = new java.util.HashMap<String, String>();
  while ((line = reader.readLine()) != null)
  {
    resultText.append(line + "\r\n");
    String[] strs = line.split(":");
    if(strs.length == 2)
    {
      keyValues.put(strs[0], strs[1]);
    }
  }

}

catch(Exception ex)
{
  throw new RuntimeException(ex);
}
finally
{
  if(reader != null)
  {
    reader.close();
  }
}

//set the download configuration to application connectionProperties
Application app = Application.getInstance();
ConnectionProperties appConnections = app.getConnectionProperties();
appConnections.setServerName(keyValues.get("server"));
appConnections.setPortNumber(Integer.parseInt(keyValues.get("port")
));
appConnections.setUrlSuffix(keyValues.get("URL Suffix"));
appConnections.setFarmId(keyValues.get("Farm ID"));

resultText.append("server name is set to: " +
appConnections.getServerName() + "\r\n");
resultText.append("server port is set to: " +
appConnections.getPortNumber() + "\r\n");
resultText.append("url suffix is set to: " +
appConnections.getUrlSuffix() + "\r\n");
```

```
resultText.append("farm id is set to: " + appConnections.getFarmId()
+ "\r\n");
```

The `Textview` output is:

```
the seed data file: data/data/com.app/files/seedData/
SUPOnboardingSeedData.txt
server: relayserver.sybase.com
port: 80
URL Suffix: /ias_relay_server/client/rs_client.dl
Farm ID: example.exampleMBS
server name is set to: relayserver.sybase.com
server port is set to: 80
url suffix is set to: /ias_relay_server/client/rs_client.dl
farm id is set to: example.exampleMBS
```

For more information on the Afaria APIs and the meanings of return codes, see the Afaria documentation.

Using Certificates from Afaria for Authentication

One of the features of Afaria is the ability to provide a device with a signed certificate that could be used as an authentication credential for SAP Mobile Platform. This note explains how to take a certificate provided by Afaria and convert it into a form suitable for use with SAP Mobile Platform.

Prerequisites:

- The application has been built using the SAP Mobile Platform framework headers and libraries.
- The application has been registered with the Afaria server as an application policy and made available to the client device.

In SAP Mobile Platform, a certificate can be used for authentication by creating a LoginCertificate object (the LoginCertificate class), and setting that as the certificate property in the client's synchronization profile.

After calling the Afaria APIs to get initial settings and configuration data, an application using Afaria may obtain a signed certificate using this API:

```
X509Certificate
com.sybase.afaria.SeedDataAPI.retrieveCertificate(RSAPublicKey
arg0, RSAPrivateKey arg1,  String arg2, String arg3,
SeedDataCredentials arg4) throws SeedDataAPIException
```

After this, the application will have an X509Certificate object. The certificate data in the X509Certificate object cannot be used as a LoginCertificate. It must be converted into a LoginCertificate.

This sample code shows how to get the Afaria certificate, create a `LoginCertificate` object, and attach it to a SAP Mobile Platform synchronization profile.

The part of the code from the top through the section which retrieves the `LoginCertificate` object is performed only once during application initialization where you are obtaining the certificate through Afaria. The `LoginCertificate` is next stored in the data vault. Each time the application runs thereafter, it retrieves the `LoginCertificate` from the data vault and sets it into the `connProperties.setLoginCertificate(lc)`; as shown, before synchronizing.

```
String commonName = "SMP-SSO";
String passWord = "smp";
String pkcsFile = "/mnt/sdcard/SMP-SSO.pfx";
//first, initialize SeedDataAPI using current Android Activity
context
SeedDataAPI.initialize(this);

//generate a key pair using java.security API
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
keyPairGen.initialize(1024);
KeyPair keyPair = keyPairGen.generateKeyPair();
RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();

//get the X509Certificate object from Afaria server by Afaria API
X509Certificate cer = SeedDataAPI.retrieveCertificate(publicKey,
privateKey, commonName, passWord, null);

//we need to wrap the X509Certificate and private key to a PKCS12
Certificate
java.security.KeyStore ks =
java.security.KeyStore.getInstance("PKCS12");
ks.load(null, passWord.toCharArray());
ks.setCertificateEntry(commonName, cer);
Certificate[] chain = {cer};
ks.setKeyEntry(commonName, privateKey, passWord.toCharArray(),
chain);
FileOutputStream out = new FileOutputStream(pkcsFile);
ks.store(out, passWord.toCharArray());

//call API to get LoginCertificate object from the PKCS12 certificate
file
LoginCertificate lc =
CertificateStore.getDefault().getSignedCertificateFromFile(pkcsFile
, passWord);

//use the loginCertificate to register Application
Application app = Application.getInstance();
ConnectionProperties connProperties = app.getConnectionProperties();
connProperties.setLoginCertificate(lc);
```

Security APIs

The security APIs allow you to customize some aspects of secure storage.

DataVault

The DataVault class provides encrypted storage of occasionally used, small pieces of data. All exceptions thrown by DataVault methods are of type DataVaultException.

By linking the `DataVaultLib.jar`, you can use the DataVault class for on-device persistent storage of certificates, database encryption keys, passwords, and other sensitive items. Use this class to

- Create a vault
- Set a vault's properties
- Store objects in a vault
- Retrieve objects from a vault
- Change the password used to access a vault

The `DataVaultLib.jar` is a standalone library using which you can consume the data vault APIs directly. To consume the data vault APIs via the SAP Mobile Platform Messaging Channel, you need to link the `ClientLib.jar` to your project.

The contents of the data vault are strongly encrypted using AES-256. The `DataVault` class allows you create a named vault, and specify a password and salt used to unlock it. The password can be of arbitrary length and can include any characters. The password and salt together generate the AES key. If the user enters the same password when unlocking, the contents are decrypted. If the user enters an incorrect password, exceptions occur. If the user enters an incorrect password a configurable number of times, the vault is deleted and any data stored within it becomes unrecoverable. The vault can also relock itself after a configurable amount of time.

Typical usage of the `DataVault` is to implement an application login screen. Upon application start, the user is prompted for a password, which unlocks the vault. If the unlock attempt is successful, the user is allowed into the rest of the application. User credentials for synchronization can also be extracted from the vault so the user need not reenter passwords.

Private Data Vault

In addition to the `DataVault` class, there is a `PrivateDataVault` class that has the features described below:

- Used to securely store data that is specific to an application.
- You do not have to install the `SybaseDataProvider.apk` to use the private data vault.

init

Initialization function that you must call with the application's context before you call any of the other vault methods. In addition to saving the context for later use, this method also initializes static member variables (such as encryption objects).

Syntax

```
public static void init(android.content.Context oContext)
```

Parameters

- **oContext –**

  ```
  Valid application context.
  ```

Returns

None.

Examples

- **Initialize**

  ```
  DataVault.init(oContext);
  ```

createVault

Creates a new secure store (a vault)

A unique name is assigned, and after creation, the vault is referenced and accessed by that name. This method also assigns a password and salt value to the vault. If a vault with the same name already exists, this method throws an exception. A newly created vault is in the unlocked state.

Syntax

```
public static DataVault createVault(
   String name,
   String password,
   String salt
)
```

Parameters

- **name –** an arbitrary name for a `DataVault` instance on this device. This name is effectively the primary key for looking up `DataVault` instances on the device, so it cannot use the same name as any existing instance. If it does, this method throws an exception with error code INVALID_ARG. The name also cannot be empty or null.

- **password –** the initial encryption password for this `DataVault`. This is the password needed for unlocking the vault. If null is passed, a default password is computed and used.
- **salt –** the encryption salt value for this `DataVault`. This value, combined with the password, creates the actual encryption key that protects the data in the vault. If null is passed, a default salt is computed and used.

Returns

Returns the newly created instance of the DataVault with the provided ID. The returned DataVault is in the unlocked state with default configuration values. To change the default configuration values, you can immediately call the "set" methods for the values you want to change.

Examples

- **Create a data vault –** creates a new data vault called `myVault`.

```
DataVault vault = null;
if (!DataVault.vaultExists("myVault"))
{
   vault = DataVault.createVault("myVault", "password", "salt");
}
else
{
   vault = DataVault.getVault("myVault");
}
```

vaultExists
Tests whether the specified vault exists.

Syntax

```
public static boolean vaultExists(String name)
```

Parameters

- **name –** the vault name.

Returns

Returns true if the vault exists; otherwise returns false.

Examples

- **Check if a data vault exists –** checks if a data vault called `myVault` exists, and if so, deletes it.

```
if (DataVault.vaultExists("myVault"))
{
   DataVault.deleteVault("myVault");
}
```

getVault
Retrieves a vault.

Syntax

```
public static DataVault getVault(String name)
```

Parameters

- **name –** the vault name.

Returns

Returns a `DataVault` instance.

If the vault does not exist, a `DataVaultException` is thrown.

deleteVault
Deletes the specified vault from on-device storage.

Note: When you have a shared data vault, if one application deletes the vault, the data vault is no longer accessible in the other application.

If the vault does not exist, this method throws an exception. The vault need not be in the unlocked state, and can be deleted even if the password is unknown.

Syntax

```
public static void deleteVault(String name)
```

Parameters

- **name –** the vault name.

Examples

- **Delete a data vault –** deletes a data vault called `myVault`.

```
if (DataVault.vaultExists("myVault"))
{
   DataVault.deleteVault("myVault");
}
```

getDataNames
Retrieves information about the data names stored in the vault.

The application can pass the data names to `getValue` or `getString` to retrieve the data values.

Syntax

```
public abstract DataVault.DVDataName[] getDataNames()
```

Parameters

None.

Returns

Returns a `DVPasswordPolicy` object, as an array of `DVDataName` structure objects.

Examples

- **Get data names**

  ```
  // Call getDataNames to retrieve all stored element names from our
  data vault.
  DataVault.DVDataName[] dataNameArray = oDataVault.getDataNames();
  for ( int i = 0; i < dataNameArray.length; i++ )
  {
    if ( dataNameArray[i].iType == DataVault.DV_DATA_TYPE_STRING )
    {
      String thisStringValue =
  oDataVault.getString( dataNameArray[i].sName );
    }
    else
    {
      byte[] thisBinaryValue =
  oDataVault.getValue( dataNameArray[i].sName );
    }
  }
  ```

setPasswordPolicy

Stores the password policy and applies it when `changePassword` is called, or when validating the password in the `unlock` method.

If the application has not set a password policy using this method, the data vault does not validate the password in the `createVault` or `changePassword` method. An exception is thrown if there is any invalid (negative) value in the `passwordPolicy` object.

Syntax

```
public abstract void setPasswordPolicy(DataVault.DVPasswordPolicy
oPasswordPolicy)
```

Parameters

- **oPasswordPolicy –** the password policy constraints.

Returns

None.

Examples

- **Set a password policy**

```
// SetPasswordPolicy() locks the vault to ensure the old password
// conforms to the new password policy settings.
oDataVault.setPasswordPolicy( oPasswordPolicy );
```

Password Policy Structure

A structure defines the policy used to generate the password.

Table 6. Password Policy Structure

Name	Type	Description
defaultPasswordAllowed	Boolean	Indicates if client application is allowed to use default password for the data Vault. If this is set to TRUE and if client application uses default password then minLength, hasDigits, hasUpper, hasLower and hasSpecial parameters in the policy are ignored.
minimumLength	Integer	The minimum length of the password.
hasDigits	Boolean	Indicates if the password must contain digits.
hasUpper	Boolean	Indicates if the password must contain uppercase characters.
hasLower	Boolean	Indicates if the password must contain lowercase characters.
hasSpecial	Boolean	Indicates if the password must contain special characters. The set of special characters is: “~!@#$%^&*()-+”.
expirationDays	Integer	Specifies password expiry days from the date of setting the password. 0 indicates no expiry.

Name	Type	Description
minUniqueChars	Integer	The minimum number of unique characters in the password. For example, if length is 5 and minUniqueChars is 4 then “aaate” or “ababa” would be invalid passwords. Instead, “aaord” would be a valid password.
lockTimeout	Integer	The timeout value (in seconds) after which the vault will be locked from the unlock time. 0 indicates no timeout. This value overrides the value set by `setLockTimeout` method.
retryLimit	Integer	The number of failed unlock attempts after which data vault is deleted. 0 indicates no retry limit. This value overrides the value set by the `setRetryLimit` method.

Settings for Password Policy

The client applications use these settings to fill the PasswordPolicy structure. The default values are used by the data vault when no policy is configured. The defaults are also used in SAP Control Center in the default template. The SAP Mobile Platform administrator can modify these settings through SAP Control Center. The application must set the password policy for the data vault with the administrative (or alternative) settings.

Note: Setting the password policy locks the vault. The password policy is enforced when `unlock` is called (because the password is not saved, calling `unlock` is the only time that the policy can be evaluated).

- **MCL_PROP_ID_PWDPOLICY_ENABLED –** Boolean property with a default value of false. Indicates if a password policy is enabled by the administrator.
- **MCL_PROP_ID_PWDPOLICY_DEFAULT_PASSWORD_ALLOWED –** Boolean property with a default value of false. Indicates if the client application is allowed to use the default password for the data vault.
- **MCL_PROP_ID_PWDPOLICY_MIN_LENGTH –** Integer property with a default value of 0. Defines the minimum length for the password.
- **MCL_PROP_ID_PWDPOLICY_HAS_DIGITS –** Boolean property with a default value of false. Indicates if the password must contain digits.

- **MCL_PROP_ID_PWDPOLICY_HAS_UPPER –** Boolean property with a default value of false. Indicates if the password must contain at least one uppercase character.
- **MCL_PROP_ID_PWDPOLICY_HAS_LOWER –** Boolean property with a default value of false. Indicates if the password must contain at least one lowercase character.
- **MCL_PROP_ID_PWDPOLICY_HAS_SPECIAL –** Boolean property with a default value of false. Indicates if the password must contain at least one special character. A special character is a character in this set "~!@#$%^&*()-+".
- **MCL_PROP_ID_PWDPOLICY_EXPIRATION_DAYS –** Integer property with a default value of 0. Specifies the number of days in which password will expire from the date of setting the password. Password expiration is checked only when the vault is unlocked.
- **MCL_PROP_ID_PWDPOLICY_MIN_UNIQUE_CHARS –** Integer property with a default value of 0. Specifies minimum number of unique characters in the password. For example, if minimum length is 5 and minUniqueChars is 4 then "aaate" or "ababa" would be invalid passwords. Instead, "aaord" would be a valid password.
- **MCL_PROP_ID_PWDPOLICY_LOCK_TIMEOUT –** Integer property with a default value of 0. Specifies timeout value (in seconds) after which the vault is locked from the unlock time. 0 indicates no timeout.
- **MCL_PROP_ID_PWDPOLICY_RETRY_LIMIT –** Integer property with a default value of 0. Specifies the number of failed unlock attempts after which data vault is deleted. 0 indicates no retry limit.

Password Errors

Password policy violations cause exceptions to be thrown.

Table 7. Password Errors

Name	Value	Description
PASSWORD_REQUIRED	50	Indicates that a blank or null password was used when the password policy does not allow default password.
PASSWORD_UN-DER_MIN_LENGTH	51	Indicates that the password length is less than the required minimum.
PASSWORD_RE-QUIRES_DIGIT	52	Indicates that the password does not contain digits.
PASSWORD_RE-QUIRES_UPPER	53	Indicates that the password does not contain upper case charac-ters.

Name	Value	Description
PASSWORD_RE-QUIRES_LOWER	54	Indicates that the password does not contain lower case characters.
PASSWORD_RE-QUIRES_SPECIAL	55	Indicates that the password does not contain one of these special characters: ~!@#$%^&*()-+.
PASSWORD_UN-DER_MIN_UNIQUE	56	Indicates that the password contains fewer than the minimum required number of unique characters.
PASSWORD_EXPIRED	57	Indicates that the password has been in use longer than the number of configured expiration days.

getPasswordPolicy android blackberry

Retrieves the password policy set by `setPasswordPolicy`.

Use this method once the `DataVault` is unlocked.

Syntax

```
public abstract DataVault.DVPasswordPolicy getPasswordPolicy()
```

Parameters

None.

Returns

Returns a `passwordPolicy` structure that contains the policy set by `setPasswordPolicy`.

Returns a `DVPasswordPolicy` object with the default values if no password policy is set.

Examples

- **Get the current password policy**

  ```
  // Call getPasswordPolicy() to return the current password policy
  settings.
        DataVault.DVPasswordPolicy oCurrentPolicy =
  oDataVault.getPasswordPolicy();
  ```

isDefaultPasswordUsed

Checks whether the default password is used by the vault.

Use this method once the `DataVault` is unlocked.

Syntax

```
public boolean isDefaultPasswordUsed()
```

Parameters

None.

Returns

Returns	Indicates
true	Both the default password and the default salt are used to encrypt the vault.
false	Either the default password or the default salt is not used to encrypt the vault.

Examples

- **Check if default password used**

  ```
  // Call isDefaultPasswordused() to see if we are using an
  automatically
  // generated password (which we are).
  boolean isDefaultPasswordUsed =
  oDataVault.isDefaultPasswordUsed();
  ```

  This code example lacks exception handling. For a code example that includes exception handling, see *Developer Guide: Android Object API Applications* > *Client Object API Usage* > *Security APIs* > *DataVault* > *Code Sample*.

lock

Locks the vault.

Once a vault is locked, you must unlock it before changing the vault's properties or storing anything in it. If the vault is already locked, `lock` has no effect.

Syntax

```
public void lock()
```

Parameters

None.

Returns

Examples

- **Locks the data vault –** prevents changing the vaults properties or stored content.

  ```
  vault.lock();
  ```

isLocked

Checks whether the vault is locked.

Syntax

```
public boolean isLocked()
```

Parameters

None.

Returns

Returns	Indicates
true	The vault is locked.
false	The vault is unlocked.

unlock

Unlocks the vault.

Unlock the vault before changing the its properties or storing anything in it. If the incorrect password or salt is used, this method throws an exception. If the number of unsuccessful attempts exceeds the retry limit, the vault is deleted.

The password is validated against the password policy if it has been set using `setPasswordPolicy`. If the password is not compatible with the password policy, an `IncompatiblePassword` exception is thrown. In that case, call `changePassword` to set a new password that is compatible with the password policy.

Syntax

```
public void unlock(String password, String salt)
```

Parameters

- **password –** the initial encryption password for this `DataVault`. If null is passed, a default password is computed and used.

- **salt –** the encryption salt value for this `DataVault`. This value, combined with the password, creates the actual encryption key that protects the data in the vault. If null is passed, a default salt is computed and used.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `INVALID_PASSWORD`.

Examples

- **Unlocks the data vault –** once the vault is unlocked, you can change its properties and stored content.

  ```
  if (vault.isLocked())
  {
     vault.unlock("password", "salt");
  }
  ```

setString

Stores a string object in the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public void setString(
   String name,
   String value
)
```

Parameters

- **name –** the name associated with the string object to be stored.
- **value –** the string object to store in the vault.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `INVALID_PASSWORD`.

Examples

- **Set a string value –** creates a test string, unlocks the vault, and sets a string value associated with the name `"testString"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

  ```
  string teststring = "ABCDEFabcdef";
  try
  ```

```
{
   vault.unlock("password", "salt");
   vault.setString("testString", teststring);
}
catch (DataVaultException e)
{
   System.out.println("Exception: " + e.toString());
}
finally
{
   vault.lock();
}
```

getString

Retrieves a string value from the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public String getString(String name)
```

Parameters

- **name –** the name associated with the string object to be retrieved.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `INVALID_PASSWORD`.

Examples

- **Get a string value –** unlocks the vault and retrieves a string value associated with the name "`testString`" in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
try
{
   vault.unlock("password", "salt");
   string retrievedstring = vault.getString("testString");
}
catch (DataVaultException e)
{
   System.out.println("Exception: " + e.toString());
}
finally
{
   vault.lock();
}
```

setValue

Stores a binary object in the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public void setValue(
   string name,
   byte[] value
)
```

Parameters

- **name –** the name associated with the binary object to be stored.
- **value –** the binary object to store in the vault.

Returns

None.

Examples

- **Set a binary value –** unlocks the vault and stores a binary value associated with the name `"testValue"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

  ```
  try
  {
     vault.unlock("password", "salt");
     vault.setValue("testValue", new byte[] { 1, 2, 3, 4, 5});
  }
  catch (DataVaultException e)
  {
     System.out.println("Exception: " + e.toString());
  }
  finally
  {
     vault.lock();
  }
  ```

getValue

Retrieves a binary object from the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public byte[] getValue(string name)
```

Parameters

- **name –** the name associated with the binary object to be retrieved.

Returns
None.

Examples

- **Get a binary value –** unlocks the vault and retrieves a binary value associated with the name `"testValue"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
try
{
   vault.unlock("password", "salt");
   byte[] retrievedvalue = vault.getValue("testValue");
}
catch (DataVaultException e)
{
   System.out.println("Exception: " + e.toString());
}
finally
{
   vault.lock();
}
```

deleteValue
Deletes the specified value.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public static void deleteValue(String name)
```

Parameters

- **name –** the name of the value to be deleted.

Returns
None.

Examples

- **Delete a value –** deletes a value called `myValue`.

```
DataVault.deleteValue("myValue");
```

changePassword (two parameters)

Changes the password for the vault. Use this method when the vault is unlocked.

Modifies all name/value pairs in the vault to be encrypted with a new password/salt. If the vault is locked or the new password is empty, an exception is thrown.

Syntax

```
public void changePassword(
   String newPassword,
   String newSalt
)
```

Parameters

- **newPassword –** the new password.
- **newSalt –** the new encryption salt value.

Returns

None.

Examples

- **Change the password for a data vault –** changes the password to `"newPassword"`. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

  ```
  try
  {
     vault.unlock("password", "salt");
     vault.changePassword("newPassword", "newSalt");
  }
  catch (DataVaultException e)
  {
     System.out.println("Exception: " + e.toString());
  }
  finally
  {
     vault.lock();
  }
  ```

changePassword (four parameters)

Changes the password for the vault. Use this method when the vault is locked

This overloaded method ensures the new password is compatible with the password policy, uses the current password to unlock the vault, and changes the password of the vault to a new password. If the current password is not valid an `InvalidPassword` exception is thrown. If the new password is not compatible with the password policy set in `setPasswordPolicy` then an `IncompatiblePassword` exception is thrown.

Syntax

```
public abstract void changePassword(string sCurrentPassword,
    string sCurrentSalt,
    string sNewPassword,
    string sNewSalt)
```

Parameters

- **currentPassword –** the current encryption password for this data vault. If a null value is passed, a default password is computed and used.
- **currentSalt –** the current encryption salt value for this data vault. If a null value is passed, a default password is computed and used.
- **newPassword –** the new encryption password for this data vault. If a null value is passed, a default password is computed and used.
- **newSalt –** the new encryption salt value for this data vault. This value, combined with the password, creates the actual encryption key that protects the data in the vault. This value may be an application-specific constant. If a null value is passed, a default password is computed and used.

Returns

None.

Examples

- **Change the password for a data vault**

  ```
  // Call changePassword with four parameters, even if the vault is
  locked.
  // Pass null for oldSalt and oldPassword if the defaults were
  used.
  oDataVault.changePassword( null, null, "password!1A",
  "saltD#ddg#k05%gnd[!1A" );
  ```

client package

Members

All public members of the client package.

- **ODPAppSettings class –** Consists of methods used to retrieve setting details required by the application.
- **ODPCertificateManager class –** Consists of methods used to provide ccess to certificate store.
- **ODPClientConnection class –** Consists of methods used for client-server connection.
- **ODPClientListeners class –** Consists of methods used to notify different types of events and respond to them.
- **ODPException class –** Consists of the exceptions thrown by the client library.

- **ODPUserManager class –** Consists of methods used to register or uregister a user.
- **SDMResponse class –** Consists of the response object received by the client after request-response onSuccess() or onError().

ODPAppSettings class

Consists of methods used to retrieve setting details required by the application.

Syntax

`public class  ODPAppSettings`

IsServerKeyProvisioned() method

Check if the public key of the SUP server is provisioned on the client.

Syntax

`boolean IsServerKeyProvisioned ()` throws `ODPException`

Returns

If the key is provisioned, the value 'true' is returned, else 'false'.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPAppSettings appSettings = new ODPAppSettings();
if(appSettings.IsServerKeyProvisioned())
{
Log.i(null,"IsServerKeyProvisioned is true");
}
```

getApplicationEndPoint() method

Retrieve the application end-point with which you can access business data.

Syntax

`String getApplicationEndPoint ()` throws `ODPException`

Returns

Application end-point

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPAppSettings odpAppSettings = new ODPAppSettings();
odpAppSettings.getApplicationEndPoint();
```

getCustomizationResourceBundle(String, String, String) method
Download the customization resources as binary data as notified using CUSTOMIZATION_RESOURCE property.

Syntax

`byte[] getCustomizationResourceBundle ( String` *CUSTOMIZATION_RESOURCE* `, String` *UserName* `, String` *Password* `)` throws `ODPException`

Parameters

- **CUSTOMIZATION_RESOURCE –** If null/empty fetches the default as assigned customization resource.
- **UserName –** User name.
- **Password –** Password.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
1. Initialize and set the listener
ODPClientConnection.initInstance(getApplicationContext(),
appName);
ODPClientConnection client= ODPClientConnection.getInstance();
configChangeListener listener=new configChangeListener();
client.addConfigurationChangeListener(listener);
2. Class the implements the IODPConfigurationChangeListener
public class configChangeListener implements
IODPConfigurationChangeListener{public void
onConfigurationChange(int key, String value) {
//Is called when there is a customization bundle added/changed
from SCC
if(key==CUSTOMIZATION_RESOURCES)
{
try {
ODPAppSettings las = new ODPAppSettings();
byte bundleData[] = null;
try {
bundleData =
las.getCustomizationResourceBundle(value,Helper.USERNAME,
Helper.PASSWORD);
} catch (ODPException e) {
e.printStackTrace();
}ZipInputStream zipStream = new ZipInputStream(new
ByteArrayInputStream(bundleData));
ZipEntry entry = null;
while ((entry = zipStream.getNextEntry()) != null)
{
String entryName = entry.getName();
zipStream.closeEntry();
Log.i("Tag", "FileName***"+entryName
+"size***"+entry.getCompressedSize());
}
zipStream.close();
}
catch (IOException e) {
throw new RuntimeException(e);
}
}
}
}
```

getFarmID() method

Retrieve the farm ID provisioned in the client repository.

Syntax

`String` `getFarmID` `()` throws `ODPException`

Returns

Farm ID

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPAppSettings odpAppSettings = new ODPAppSettings();
odpAppSettings.getFarmID();
```

getPasswordPolicy() method

Retrieve the password policy obtained from the SUP server.

Syntax

`DataVault.DVPasswordPolicy` `getPasswordPolicy` `()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPAppSettings odpAppSettings = new ODPAppSettings();
odpAppSettings.getPasswordPolicy();
```

getPortNumber() method

Retrieve the server port number provisioned in the client repository.

Syntax

`int getPortNumber ()` throws `ODPException`

Returns

Port number

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPAppSettings odpAppSettings = new ODPAppSettings();
int portNo = odpAppSettings.getPortNumber();
```

getPushEndPoint() method

Retrieve the push end-point which is used by the application to specify the delivery address in the subscription request for notification payload.

Syntax

`String getPushEndPoint ()` throws `ODPException`

Returns

Push end-point

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPAppSettings odpAppSettings = new ODPAppSettings();
odpAppSettings.getPushEndPoint();
```

getServer() method

Retrieve the SUP server name provisioned in the client repository.

Syntax

`String` `getServer ()` throws `ODPException`

Returns

SUP server name

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPAppSettings odpAppSettings = new ODPAppSettings();
String ServerName = odpAppSettings.getServer();
```

log variable

Syntax

`ODPLogger  log`

ODPCertificateManager class

Consists of methods used to provide ccess to certificate store.

Syntax

`public class  ODPCertificateManager`

ODPCertInfo class

Syntax

`public class  ODPCertInfo`

ODPCertInfo(CertInfo) constructor

Syntax

`ODPCertInfo ( CertInfo` *certInfo* `)`

certInfo variable

Syntax
`CertInfo certInfo`

getCertificateDisplayName() method
Display the name of the certificate.

Syntax
`String getCertificateDisplayName ()`

Returns
Certificate name list

getIssuer() method
Retrieve the certificate issue authority information.

Syntax
`String getIssuer ()`

Returns
Distinguish format name for certificate issue authority

getIssuerCN() method
Retrieve the common name for certificate issue authority.

Syntax
`String getIssuerCN ()`

Returns
Name of the certificate issue authority

getSubject() method
Retrieves the subject of certificate in distinguished name format.

Syntax
`String getSubject ()`

Returns
Subject of the certificate

getSubjectCN() method

Retrieves the subject of certificate common name.

Syntax

`String getSubjectCN ()`

Returns

Subject of the certificate common name

getValidityBeginDate() method

Retrieve the validity start date of the certificate.

Syntax

`Date getValidityBeginDate ()`

Returns

Validity start date

getValidityExpiryDate() method

Retrieve the validity expiry date of the certificate.

Syntax

`Date getValidityExpiryDate ()`

Returns

Validity end date

getSignedCertificateFromFile(String, String) method

Returns the requested certificate from the file as a base64 encoded string.

Syntax

`String getSignedCertificateFromFile ( String` *certificatePath* `, String` *certificatePassword* `)`

Parameters

- **certificatePath –** Path of the certificate on the file system.
- **certificatePassword –** Password to read the certificate.

Returns

Certificate

Examples

- **Example 1**

```
String certPass =
ODPCertificateManager.getSignedCertificateFromFile("/mnt/sdcard/
SUPUSER.p12", "mobile").toString();
```

ODPClientConnection class
Consists of methods used for client-server connection.

Syntax
```
public class  ODPClientConnection
```

C2dmNotificationListener class

Syntax
```
package class  C2dmNotificationListener
```

onPushNotification(Hashtable) method

Syntax
`int onPushNotification ( Hashtable` *arg0* `)`

setListner(ODPClientListeners.IODPPushNotificationListener) method

Syntax
`void setListner`
`( ODPClientListeners.IODPPushNotificationListener` *sc2dml* `)`

LiteConfigurationChangeListener class

Syntax
```
package class  LiteConfigurationChangeListener
```

LiteConfigurationChangeListener() constructor

Syntax
```
LiteConfigurationChangeListener ()
```

onConfigurationChange(int, Object) method

Syntax
`void onConfigurationChange ( int` *iPropertyID* `,  Object` *oValue* `)`

setListner(ODPClientListeners.IODPConfigurationChangeListener) method

Syntax

`void setListner`
`( ODPClientListeners.IODPConfigurationChangeListener` *slcl* `)`

LiteConnectionStateListener class

Syntax

`package class  LiteConnectionStateListener`

onConnectionStateChanged(eConnectionStatus, int, int, String) method

Syntax

`void onConnectionStateChanged ( eConnectionStatus` *arg0* `,  int` *arg1* `,`
`int` *arg2* `,  String` *arg3* `)`

ODPCertificateChallengeListener class

Syntax

`package class  ODPCertificateChallengeListener`

ODPCertificateChallengeListener() constructor

Syntax

`ODPCertificateChallengeListener ()`

isServerTrusted(SSOCertManager.CertInfo[]) method
Called when server has to be trusted.

Syntax

`boolean isServerTrusted ( SSOCertManager.CertInfo[]` *aoChain* `)`

Examples

- **Example 1**

```
if(isServerTrusted(certChain)
{Log.i("Tag","Server is Trusted");}
```

setCertificateChallengeListener(IODPCertificateChallengeListener) method

Syntax

```
void setCertificateChallengeListener
( IODPCertificateChallengeListener c1c1 )
```

ODPHTTPAuthChallengeListener class

Syntax

```
package class ODPHTTPAuthChallengeListener
```

ODPHTTPAuthChallengeCredentials() method

Syntax

```
void ODPHTTPAuthChallengeCredentials ()
```

getCredentials(String, String, String) method

Syntax

```
HTTPAuthChallengeCredentials getCredentials ( String arg0 ,
String arg1 ,  String arg2 )
```

ODPHTTPErrorListener class

Syntax

```
package class ODPHTTPErrorListener
```

ODPHTTPErrorListener() constructor

Syntax

```
ODPHTTPErrorListener ()
```

onHTTPError(int, String, Hashtable) method

Syntax

```
void onHTTPError ( int arg0 ,  String arg1 ,  Hashtable arg2 )
```

ODPRequestInfoListener class

Syntax

```
package class ODPRequestInfoListener
```

BeginRequest(RequestInfo) method

Syntax
`void BeginRequest ( RequestInfo` *arg0* `)`

BeginResponse(RequestInfo) method

Syntax
`void BeginResponse ( RequestInfo` *arg0* `)`

EndRequest(RequestInfo) method

Syntax
`void EndRequest ( RequestInfo` *arg0* `)`

EndResponse(RequestInfo) method

Syntax
`void EndResponse ( RequestInfo` *arg0* `)`

Interrupted(RequestInfo, Exception) method

Syntax
`void Interrupted ( RequestInfo` *arg0* `, Exception` *arg1* `)`

TrcLvl_HIGH variable

Syntax
`final int TrcLvl_HIGH`

TrcLvl_LOW variable

Syntax
`final int TrcLvl_LOW`

TrcLvl_MEDIUM variable

Syntax
`final int TrcLvl_MEDIUM`

TrcLvl_NONE variable

Syntax
`final int TrcLvl_NONE`

addConfigurationChangeListener(ODPClientListeners.IODPConfigurationChangeListener) method
Implement IODPConfigurationChangeListener to receive push or application end-point changes.

Syntax

`void addConfigurationChangeListener ( ODPClientListeners.IODPConfigurationChangeListener` *oListener* `)` throws `ODPException`

Parameters

- **oListener –** Object that implements the ODPClientListeners.IODPConfigurationChangeListener interface.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
IODPConfigurationChangeListener configListener = new
ODPConfigurationChangeListenerImpl();
ocl.addConfigurationChangeListener(configListener);
```

Usage

To consume updates when there are changes in the Proxy settings, the application registers a listener object.The client SDK notifies this listener object whenever there is a settings update from the server. The listener object should implement the ODPClientListeners.IODPConfigurationChangeListener interface.

clearServerVerificationKey() method
Clear the SUP settings.

Syntax

`void clearServerVerificationKey ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ocl.clearServerVerificationKey();
```

connectNow() method

Syntax

`void connectNow ()` throws `ODPException`

getAppName() method

Return the application name passed during initializing the ODPClientConnection.

Syntax

`String getAppName ()`

Returns

Application name

getConfigProperty(int) method

Return the configuration property for the requested key.

Syntax

`Object getConfigProperty ( int` *key* `)` throws `ODPException`

Parameters

- **key –** Requested key.

Returns

Configuration property

Exceptions

- **ODPException class –**

getDeviceID() method

Syntax

`String getDeviceID ()`

getInstance() method

Retrieve the instance of ODPCLientConnection.

Syntax

`ODPClientConnection getInstance ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection.initInstance(getApplicationContext(),
"MyApp");
ODPClientConnection.getInstance();
```

getsUpa() method

Syntax

`String getsUpa ()`

initInstance(Context, String) method

Initialize ODPClientConnection with application context and the application name.

Syntax

`void initInstance ( Context` *context* `, String` *appID* `)` throws `ODPException`

Parameters

- **context –** Application context.
- **appID –** Application name.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection.initInstance(getApplicationContext(),
"MyApp");
```

isConnected() method

Syntax

`boolean isConnected ()`

registerForNativePush(ODPClientListeners.IODPPushNotificationListener) method

Add the implementation of IODPPushNotificationListener to receive GCM push notification.

Syntax

`void registerForNativePush (ODPClientListeners.IODPPushNotificationListener` *oListener* `)` throws `ODPException`

Parameters

- **oListener –** Instance of ODPClientListeners.IODPPushNotificationListener.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
IODPPushNotificationListener pushListener = new
ODPPushNotificationListenerImpl();
ocl.addGCMmListener(pushListener);
```

registerForPayloadPush(ISDMNetListener) method

Register to ISDMNetListener to receive payload push notification.

Syntax

`void registerForPayloadPush ( ISDMNetListener` *push* `)`

Parameters

- **push –** Instance of ISDMNetListener.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ISDMNetListener pushListener = new SDMNetListenerImpl();
ocl.registerForPayloadPush(pushListener);
```

restartClient() method

Syntax

`void restartClient ()` throws `ODPException`

resumeConnection() method

Syntax

`void resumeConnection ()` throws `ODPException`

setConfigProperty(int, Object) method

Syntax

`void setConfigProperty ( int` *arg0* `, Object` *arg1* `)` throws `ODPException`

setConnectionProfileFromFile(String) method

Syntax

`void setConnectionProfileFromFile ( String` *FilePath* `)` throws `ODPException`

setConnectionProperties(String, int, String, String, String) method

Syntax

`void setConnectionProperties ( String` *host* `, int` *port* `, String` *farmID* `, String` *username* `, String` *activationcode* `)` throws `ODPException`

setEndpointExchangeComplete(boolean) method

Syntax

`void setEndpointExchangeComplete ( boolean` *val* `)` *setHTTPHeaders(Hashtable, Hashtable) method*

Syntax

`void setHTTPHeaders ( Hashtable` *oHeaders* `, Hashtable` *oCookies* `)` throws `ODPException`

setODPCertificateChallengeListener(IODPCertificateChallengeListener) method

Set the IODPCertificateChallengeListener to receive certificate requests.

Syntax

`void setODPCertificateChallengeListener ( IODPCertificateChallengeListener` *oListerner* `)` throws `ODPException`

Parameters

- **oListerner –** Instance of IODPCertificateChallengeListener.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
IODPCertificateChallengeListener certListener = new
ODPCertificateChallengeListenerImpl();
ocl.setODPCertificateChallengeListener( certListener );
1. Implement the listener
public class ODPCertificateListener implements
IODPCertificateChallengeListener
{
// callback method for certificate verification
public boolean isServerTrusted(ODPCertInfo[] certCredentials)
{
Log.i("Do you trust the server for
credentials:"+certCredentials.toString()); //display these
credentials for the server
return true; // return true or false depending on userinput from
the UI
}
}
2. Register a listener to verify server certificate
ODPCertificateListener odpCertificateListener = new
ODPCertificateListener();
```

```
ODPUserManager.getInstance().setODPCertificateChallengeListener(o
dpCertificateListener);
```

Usage

When a client attempts to connect to a host (like a relay server), this connection is authenticated with a basic authentication challenge. To verify if the server certificate is trusted or not, the application registers a listener object with the OData SDK. If the server certificate is trusted, the connection is successful. If the server certificate is not trusted, it is rejected and the connection fails. See "Configuring Unwired Server to use Relay Server" in SAP Control Center for SAP Mobile Platform for information on the configuration of relay server properties.

setODPHTTPAuthChallengeListener(IODPHTTPAuthChallengeListener) method

Set the IODPHTTPAuthChallengeListener to receive authentication challenge.

Syntax

`void` `setODPHTTPAuthChallengeListener`
`(` `IODPHTTPAuthChallengeListener` *olistener* `)` throws `ODPException`

Parameters

- **oListerner –** Instance of IODPHTTPAuthChallengeListener.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
IODPHTTPAuthChallengeListener httpListener = new
ODPHTTPAuthChallengeListenerImpl();
ocl.setODPHTTPAuthChallengeListener( httpListener );
Register a Listener for HTTP(s) Auth Challenge
ODPCredentials odpListener = new ODPCredentials();
ODPClientConnection.setODPHTTPAuthChallengeListener(odpListener);
Implement the listenerpublic class UserRegistration{
public void startUserRegistration() {
ODPUserManager.initialize(appID);
ODPUserManager.setConnectionProfile(serverIP, serverPort,farmID);
ODPUserManager.enableHTTPS(true);
ODPCredentials odpListener = new ODPCredentials();
ODPClientConnection.setODPHTTPAuthChallengeListener(odpListener);
ODPUserManager.registerUser(username, securityConfig,password);
}
final static class ODPCredentials implements
IODPHTTPAuthChallengeListener
{
// callback method for HTTP authentication 401 challengepublic
ODPHTTPAuthChallengeCredentials getCredentials(String sHostName,
String sOldUserName, String sRealm)
{
Log.i("Get Credentials for host"+sHostName);
ODPHTTPAuthChallengeCredentials odpCredentials = new
ODPHTTPAuthChallengeCredentials(userName,pwd);
return odpCredentials;
}
}}
```

setODPHTTPErrorListener(IODPHttpErrorListener) method

Set the IODPHttpErrorListener to receive HTTP error from the network edge.

Syntax

`void` `setODPHTTPErrorListener ( IODPHttpErrorListener` *oListener* `)`
throws `ODPException`

Parameters

- **oListerner –** - Instance of IODPHttpErrorListener.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
IODPHttpErrorListener httpErrorListener = new
ODPHttpErrorListenerImpl();
ocl.setODPHTTPErrorListener( httpErrorListener );
Implement the listener
public class UserRegistration{
public void startUserRegistration() {
UserManager.initialize(appID);
UserManager.setConnectionProfile(serverIP, serverPort,farmID);
UserManager.enableHTTPS(true);
ODPErrorListener odpErrorListener = new ODPErrorListener();
ODPClientConnection.setODPHTTPErrorListener(odpErrorListener);
UserManager.registerUser(username, securityConfig,password);
}public class ODPErrorListener implements IODPHttpErrorListener
{
// callback method for HTTP Error Code Listenerpublic void
onHttpError(int iErrorCode, String sErrorMessage,Hashtable
oHeaders)
{
Log.i("Error info" +iErrorCode+sErrorMessage);
}
}
```

Usage

To ensure that OData android clients are notified of HTTP errors while establishing a connection with the network edge, implement this listener.

setRegistrationGoingOn(boolean) method

Syntax

`void setRegistrationGoingOn ( boolean` *val* `)`

setSettingExchangeComplete(boolean) method

Syntax

`void setSettingExchangeComplete ( boolean` *val* `)`

setTraceLevel(int) method

Start the end to end trace level.

Syntax

`void setTraceLevel ( int` *level* `)`

Parameters

- **level –** Trace level - TrcLvl_HIGH, TrcLvl_MEDIUM, TrcLvl_LOW and TrcLvl_NONE.

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ocl.setTraceLevel(ocl.TrcLvl_HIGH);
```

setsUpa(String) method

Syntax

`void setsUpa ( String` *sUpa* `)`

startClient() method

Start the ODPClient.

Syntax

`void startClient ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ocl.startClient();
```

startTrace() method

Start tracing the message end to end.

Syntax

`void startTrace ()`

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ocl.startTrace();
```

Usage

Check if end to end tracing is enabled before start tracing the message.

stopClient() method

Stop the ODPClient.

Syntax

`void stopClient ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ocl.stopClient();
```

stopTrace() method

Disable end to end tracing before stop tracing the message.

Syntax

`void stopTrace ()`

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ocl.stopTrace();
```

suspendConnection() method

Syntax

`void suspendConnection ()` throws `ODPException`

uploadTrace() method

Upload the BTX file to solution manager for end to end tracing.

Syntax

`void uploadTrace ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection ocl = ODPClientConnection.getInstance();
ocl.uploadTrace();
```

ODPClientListeners class

Consists of methods used to notify different types of events and respond to them.

Syntax

```
public class  ODPClientListeners
```

ODPHTTPAuthChallengeCredentials class

Consists of methods to implement the HTTP authentication challenge listener to retrieve HTTP authentication challenge credentials.

Syntax

```
public class  ODPHTTPAuthChallengeCredentials
```

ODPHTTPAuthChallengeCredentials(String, String) constructor

Syntax

`ODPHTTPAuthChallengeCredentials ( String` *sUserName* `,  String` *sPassword* `)`

sUserName variable

Syntax

```
String sUserName
```

spassword variable

Syntax

```
String spassword
```

IODPCertificateChallengeListener interface

Implement the certificate challenge listener to retrieve the authentication challenge.

Syntax

```
public interface  IODPCertificateChallengeListener
```

isServerTrusted(ODPCertInfo[]) method

Verify the server certificate required by the operating system (OS) to check the successful connection.

Syntax

`boolean isServerTrusted ( ODPCertInfo[]` *aoChain* `)`

Parameters

- **aoChain –** Certificate chain provided by the OS that it has received from the server.

IODPConfigurationChangeListener interface

Implement the configuration change listener to retrieve the events when there is any change in push or application end-point.

Syntax

```
public interface  IODPConfigurationChangeListener
```

CUSTOMIZATION_RESOURCES variable

Syntax

```
int CUSTOMIZATION_RESOURCES
```

PROXY_APPLICATION_ENDPOINT variable

Syntax

```
int PROXY_APPLICATION_ENDPOINT
```

PROXY_PUSH_ENDPOINT variable

Syntax

```
int PROXY_PUSH_ENDPOINT
```

PWDPOLICY_CHANGED variable

Syntax

```
int PWDPOLICY_CHANGED
```

PWDPOLICY_DEFAULT_PASSWORD_ALLOWED variable

Syntax

```
int PWDPOLICY_DEFAULT_PASSWORD_ALLOWED
```

PWDPOLICY_ENABLED variable

Syntax

```
int PWDPOLICY_ENABLED
```

PWDPOLICY_EXPIRES_IN_N_DAYS variable

Syntax

```
int PWDPOLICY_EXPIRES_IN_N_DAYS
```

PWDPOLICY_HAS_DIGITS variable

Syntax

`int PWDPOLICY_HAS_DIGITS`

PWDPOLICY_HAS_LOWER variable

Syntax

`int PWDPOLICY_HAS_LOWER`

PWDPOLICY_HAS_SPECIAL variable

Syntax

`int PWDPOLICY_HAS_SPECIAL`

PWDPOLICY_HAS_UPPER variable

Syntax

`int PWDPOLICY_HAS_UPPER`

PWDPOLICY_LENGTH variable

Syntax

`int PWDPOLICY_LENGTH`

PWDPOLICY_LOCK_TIMEOUT variable

Syntax

`int PWDPOLICY_LOCK_TIMEOUT`

PWDPOLICY_MIN_UNIQUE_CHARS variable

Syntax

`int PWDPOLICY_MIN_UNIQUE_CHARS`

PWDPOLICY_RETRY_LIMIT variable

Syntax

`int PWDPOLICY_RETRY_LIMIT`

onConfigurationChange(int, String) method
Notify the changes in push and application end-point.

Syntax

`void onConfigurationChange ( int` *key* `, String` *value* `)`

Parameters

- **key –** Key of the changed end-point.
- **value –** New value of the changed end-point.

IODPHTTPAuthChallengeListener interface
Implement the listener to retrieve the HTTP authentication challenge.

Syntax

```
public interface IODPHTTPAuthChallengeListener
```

getCredentials(String, String, String) method
Receive an HTTP_AUTH_FAILURE (HTTP response code 401) challenge by connection.

Syntax

`ODPHTTPAuthChallengeCredentials getCredentials ( String` *sHostName* `, String` *sOldUserName* `, String` *sRealm* `)`

Parameters

- **sHostName –** Host name of the server.
- **sOldUserName –** User name last used, which will be empty string on the first callback for the current run of the application.
- **sRealm –** Realm specified in the 401 challenge headers received from the server, if available. If the realm is not available in the response, an empty string will be passed.

Returns

ODPHTTPAuthChallengeCredentials

IODPHttpErrorListener interface
Implement the listener to retrieve the HTTP errors.

Syntax

```
public interface IODPHttpErrorListener
```

onHttpError(int, String, Hashtable) method
Receive the HTTP error when the client connects to the reverse proxy/relay server via HTTP channel.

Syntax
`void onHttpError ( int` *errorCode* `, String` *errorMessage* `, Hashtable` *httpHeaders* `)`

Parameters

- **errorCode –** Error code.
- **errorMessage –** Error message.
- **httpHeaders –** Response headers with error message.

Usage

While establishing the connection, onHTTPError listener notifies the error to the client. In such cases, the client has to implement this protocol to get notified.

IODPPushNotificationListener interface
Implement the GCM push notification listener.

Syntax
`public interface IODPPushNotificationListener`

Remarks
This method should be implemented by the application to get GCM push notification. Additionally, the implemented class needs to register this listener using: ODPClientConnection.getInstance().addGCMListener(IODPPushNotificationListener);

```
public class ApplicationPage implements
IODPPushNotificationListener{
    public int onGCMNotification( Hashtable hashValues ){}
}
```

onGCMNotification(Hashtable) method
Receive the GCM push notification.

Syntax
`int onGCMNotification ( Hashtable` *hashValues* `)`

Parameters

- **hashValues –** Key or value data pairs of GCM push notification.

Returns

1 if the payload push is retrieved from the server else it returns 0

Examples

- **Example 1**

```
public class ApplicationPage implements
IODPPushNotificationListener{
public int onGCMNotification(Hashtable hashValues){
Log.i("Received GCM notification");
return 0;
}
}
```

IODPUserRegistrationListener interface

Implement the asynchronous user registration to retrieve the registration status.

Syntax

`public interface  IODPUserRegistrationListener`

State() enumeration

Enum Constant Summary

- **SUCCESS –**
- **FAILURE –**

onAsyncRegistrationResult(State, int, String) method

Syntax

`Void onAsyncRegistrationResult ( State` *registrationState* `,  int` *errCode* `, String` *errMsg* `)`

ODPException class

Consists of the exceptions thrown by the client library.

Syntax

`public class  ODPException`

ANY_INPUT_FIELD_NULL variable
User registration fields are null.

Syntax
`final int ANY_INPUT_FIELD_NULL`

APPLICATION_ID_NULL variable
Application is not initialized.

Syntax
`final int APPLICATION_ID_NULL`

APPLICATION_USER_ALREADY_REGISTERED variable
Application user is already registered.

Syntax
`final int APPLICATION_USER_ALREADY_REGISTERED`

APPLICATION_USER_NOT_REGISTERED variable
Application user is unregistered.

Syntax
`final int APPLICATION_USER_NOT_REGISTERED`

EMPTY_RESPONSE_FROM_SERVER variable
Empty response received from server.

Syntax
`final int EMPTY_RESPONSE_FROM_SERVER`

INVALID_MODULE_ID variable
Invalid module handle ID.

Syntax
`final int INVALID_MODULE_ID`

JSON_PARSING_FAILED variable
Internal error, JSON parsing is failed.

Syntax
`final int JSON_PARSING_FAILED`

ODPException(int) constructor
UserManagerException with specified error code is constructed.

Syntax
`ODPException ( int` *errorCode* `)`

ODPException(String) constructor
UserManagerException with specified error message is constructed.

Syntax
`ODPException ( String` *message* `)`

ODPException(int, String) constructor
UserManagerException with specified error code and error message is constructed.

Syntax
`ODPException ( int` *errorCode* `,  String` *message* `)`

REGISTRATION_FAILED_UNKNOWN_ERROR variable
User registration timed out.

Syntax
```
final int REGISTRATION_FAILED_UNKNOWN_ERROR
```

REGISTRATION_LISTENER_NULL variable
Asynchronous user registration listener is unregistered with UserManager.

Syntax
```
final int REGISTRATION_LISTENER_NULL
```

SINGLETON_INITIALIZATION_FAILED variable
Singleton initialization failed.

Syntax
```
final int SINGLETON_INITIALIZATION_FAILED
```

SUP_INTERNAL_PARSING_ERROR variable
SUP internal parsing error.

Syntax
```
final int SUP_INTERNAL_PARSING_ERROR
```

getErrorCode() method
Error code of the UserManagerException is returned.

Syntax
`int getErrorCode ()`

toString() method

Syntax
`String toString ()`

ODPUserManager class
Consists of methods used to register or uregister a user.

Syntax
```
public class  ODPUserManager
```

Remarks
Additionally, there are APIs used to provision settings on the client. An application developer can later register a user using these settings.

asyncUserRegistration class

Syntax
```
private class  asyncUserRegistration
```

run() method

Syntax
`void run ()`

UserRegistered(boolean) method

Syntax
`void UserRegistered ( boolean` *val* `)`

deleteUser() method
Delete a registered application connection.

Syntax
`void deleteUser ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
lurm.deleteUser;
```

enableHTTPS(boolean) method

Enable the HTTPS connection between the client and SUP server.

Syntax

`void enableHTTPS ( boolean` *useHTTPS* `)` throws `ODPException`

Parameters

- **useHTTPS –** Flag to set HTTPS connection. Set to "true", if the protocol to be used is HTTPS. Set to "false", if the protocol to be used is HTTP. Note: The default protocol is HTTP.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.enableHTTPS(true);
```

getFarmID() method

Syntax

`String getFarmID ()`

getHost() method

Syntax

`String getHost ()`

getInstance() method

Gives back an instance of ODPUserManager.

Syntax

`ODPUserManager  getInstance ()` throws `ODPException`

Returns

Instance of ODPUserManager

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
```

getPassword() method

Syntax

`String getPassword ()`

getPort() method

Syntax

`int getPort ()`

getSecurityConfig() method

Syntax

`String getSecurityConfig ()`

getUserName() method

Syntax

`String getUserName ()`

initInstance(Context, String) method

Initialize the ODPUserManager.

Syntax

`void initInstance ( Context` *context* `, String` *appID* `)` throws `ODPException`

Parameters

- **context –** Application context.
- **appID –** Application identifier.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(getApplicationContext(), "MyApp");
```

isUserRegistered() method

Return true if user is already registered else return false.

Syntax

`boolean isUserRegistered ()`

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
if(lurm.isUserRegistered() != true){
lurm.registerUser("appUser", "SSO","appPass", true);
}
```

registerUser(String, String, String, boolean) method

Register the user using SSO2 cookie or an X.509 certificate.

Syntax

`void registerUser ( String` *username* `, String` *securityConfig* `, String` *password* `, boolean` *isSynchronous* `)` throws `ODPException`

Parameters

- **username –** Valid user name.
- **securityConfig –** - Security configuration for the registered application provided by the administrator in SCC.
- **password –** - Password. In case of SSO2 cookie this field will be the standard password used to authenticate the user. In case X509 certificates are used, the password field will contain the Base64 encoded string of the certificate binary.
- **isSynchronous –** - flag to set the registration as synchronous or asynchronous.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
lurm.registerUser("appUser", "SSO","appPass", true);
```

registerUser(String, String, boolean) method

Authenticate the device using the username and activation code which is specified while creating the user in SCC.

Syntax

`void registerUser ( String` *username* `,  String` *activationCode* `,  boolean` *isSynchronous* `)` throws `ODPException`

Parameters

- **username –** User name.
- **activationCode –** Activation code set in SCC.
- **isSynchronous –** Flag to set the registration as synchronous or asynchronous.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
lurm.registerUser("appUser", "123", true);
```

setConnectionProfile(String, int, String) method

Specify the connectivity details of the SUP server.

Syntax

`void setConnectionProfile ( String` *host* `, int` *port* `, String` *farmID* `)` throws `ODPException`

Parameters

- **host –** IP address of the SUP server.
- **port –** Port of the server.
- **farmID –** Company name or identifier.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
lurm.setConnectionProfile("http://supServer.com", 5001, appFarm)
```

setConnectionProfileFromFile(String) method

Specify the connectivity details of the SUP server.

Syntax

`void setConnectionProfileFromFile ( String` *FilePath* `)` throws `ODPException`

Parameters

- **filePath –** Path to the file containing connectivity details.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
lurm.setConnectionProfileFromFile("/mnt/sdcard/connDetails.txt")
```

setHTTPHeaders(Hashtable, Hashtable) method

Enable the users to set the HTTP headers and cookies.

Syntax

`void setHTTPHeaders ( Hashtable` *oHeaders* `, Hashtable` *oCookies* `)` throws `ODPException`

Parameters

- **oHeaders –** Hashtable for HTTP headers as key value pairs.
- **oCookies –** Hashtable for HTTP cookies as key value pairs.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
Hashtable<String, String> oHeaders = new Hashtable<String,
String>();
Hashtable<String, String> oCookies = new Hashtable<String,
String>();
oHeaders.put("Authorization", "Basic
c3liYXNlMTAxOnN5YmFzZTEyMw==");
oCookies.put("Cookie", "gkRrDBtZ");
ODPUserManager.setHTTPHeaders(oHeaders, oCookies);
```

setRelayUrlTemplate(String) method

Enable the users to set the relay server url template if it is different from the standard template.

Syntax

`void setRelayUrlTemplate ( String` *urlSuffix* `)` throws `ODPException`

Parameters

- **urlSuffix –** Relay server URL template suffix.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager  lurm = ODPUserManager.getInstance();
lurm.setRelayUrlTemplate("myURLTemplate");
```

setUserRegistrationListener(ODPClientListeners.IODPUserRegistrationListener) method

Set the listener object to get notification when the user is registered.

Syntax

`void setUserRegistrationListener ( ODPClientListeners.IODPUserRegistrationListener` *registrationListener* `)`

Parameters

- **registrationListener –** Object implements the IUserRegistrationListener interface to notify the user registration.

SDMResponse class

Consists of the response object received by the client after request-response onSuccess() or onError().

Syntax

```
public class  SDMResponse
```

SDMResponse(StatusLine) constructor

Syntax

`SDMResponse ( StatusLine` *statusline* `)`

getBinaryToBeStreamed() method

Syntax

```
MoBinary getBinaryToBeStreamed ()
```

getCookie() method

Client to retrieve the cookie required for asynchronous streaming.

Syntax

```
String getCookie ()
```

Returns
cookie

Examples

- **Example 1**

```
SDMResponse resp = (SDMResponse)response;
resp.getCookie();
```

getCurrentOffset() method
Client retrieves the current offset required for asynchronous streaming.

Syntax
`int getCurrentOffset ()`

Returns
offset value

Examples

- **Example 1**

```
SDMResponse resp = (SDMResponse)response;
resp.getCurrentOffset();
```

getDataStream(byte[], int) method
Stream the data synchronously.

Syntax
`int getDataStream ( byte[]` *buffer* `,  int` *chunkSize* `)`

Parameters

- **buffer –** Stores the data
- **chunkSize –** Size of data in chunks

Returns
bytesStreamed

Examples

- **Example 1**

```
SDMResponse resp = (SDMResponse)response;
while((bytesRecvd = resp.getDataStream(buf,chunk_size))>0)
{
```

```
...
}
```

getDataStream(byte[], int, int, String) method
Stream the data asynchronously in case of data streaming.

Syntax

`int getDataStream ( byte[]` *buffer* `,  int` *chunkSize* `,  int` *offset* `,  String` *cookie* `)`

Parameters

- **buffer –**
- **chunkSize –**
- **offset –**
- **cookie –**

Returns

byteStreamed

Examples

- **Example 1**

```
SDMResponse resp = (SDMResponse)response;
while((bytesRecvd =
resp.getDataStream(buf,chunk_size,offset,cookie))>0)
{
...
}
```

getHeaderOffset() method

Syntax

`int getHeaderOffset ()`

isDataAvailable() method
Client to check if the data needs to be streamed or not.

Syntax

`boolean isDataAvailable ()`

Returns

isDataAvailable

Examples

- **Example 1**

```
SDMResponse resp = (SDMResponse)response;
resp.isDataAvailable();
```

setBinaryToBeStreamed(MoBinary) method

Syntax

`void setBinaryToBeStreamed ( MoBinary` *binaryToBeStreamed* `)`

setHeaderOffset(int) method

Syntax

`void setHeaderOffset ( int` *headerOffset* `)`

CHAPTER 4 Developing BlackBerry Applications

Provides information about using advanced SAP® Mobile Platform features to create applications for RIM BlackBerry devices. The audience is advanced developers who are familiar working with APIs, but who may be new to SAP Mobile Platform.

Using Online Data Proxy, you can connect a device to an OData-based back-end system. All Online Data Proxy client libraries provide secure communication to the SAP Mobile Platform server in addition to parsing, caching, persistence, connectivity, supportability and secure storage.

Describes requirements for developing a device application for the platform. Also included are task flows for the development options, procedures for setting up the development environment and API references.

1. *Getting Started Task Flow for BlackBerry Applications*

   This task flow describes requirements for developing a device application for the platform. It includes task flows for the development options, procedures for setting up the development environment, and API documentation.
2. *Development Task Flow for BlackBerry Applications*

   For an application to work in an ODP scenario, it needs to be initialized first. Additionally, an application developer uses APIs specific to ODP that enables the application to send and receive data.
3. *Deploying Applications to Devices*

   This section describes how to deploy customized mobile applications to devices.
4. *OData SDK Components and APIs*

   The OData SDK for BlackBerry provides the means to easily build an application which relies on the OData protocol and its additions made by SAP.
5. *ODP SDK API Usage*

   The ODP SDK consists of APIs used to customize applications to send and receive data using Online Data Proxy.

Getting Started Task Flow for BlackBerry Applications

This task flow describes requirements for developing a device application for the platform. It includes task flows for the development options, procedures for setting up the development environment, and API documentation.

See also

- *Development Task Flow for BlackBerry Applications* on page 317

Configuring the BlackBerry Developer Environment

This section describes how to set up your BlackBerry development environment and provides the location of required JAR files.

Installing the BlackBerry Development Environment

Download and install either the BlackBerry JDE or the BlackBerry Java plug-in for Eclipse (eJDE).

For information on transitioning from the BlackBerry JDE to the eJDE, view the video at the Research In Motion Developer Video Library Web site: *http://supportforums.blackberry.com/t5/Java-Development/tkb-p/java_dev%40tkb?labels=video*

Installing the BlackBerry Java Plug-in for Eclipse

The BlackBerry Java Plug-in for Eclipse is an IDE for developing BlackBerry applications.

Prerequisites

You must have a BlackBerry developer account to download the BlackBerry Java Plug-in for Eclipse. You may be required to register if you do not already have an account.

Task

1. Go to *http://us.blackberry.com/developers/javaappdev/* and download the BlackBerry Java Plug-in for Eclipse (full installer) to a temporary folder.
2. Double-click the setup application file.
3. Click **Run**.
4. On the Introduction page, click **Next**.
5. Accept the terms of the license agreement and click **Next**.
6. Create and select a new, empty folder for the installation directory and click **Next**.
7. Review the information on the Pre-installation Summary screen and click **Install**.
8. Click **Done**.
   The installation is complete.
9. (Optional). Copy the `plugin` and `features` folders from the installation to `SMP_HOME\Unwired_WorkSpace\Eclipse\sybase_workspace\mobile\eclipse`.
   This step ensures that SAP Mobile WorkSpace contains the BlackBerry Java Plug-in for Eclipse, and that users can directly use it from SAP Mobile WorkSpace instead of opening another instance of Eclipse to work with the BlackBerry Java Plug-in for Eclipse.

Downloading the BlackBerry JDE

To generate and distribute BlackBerry device applications, download the BlackBerry JDE and its prerequisites from the BlackBerry Web site.

Prerequisites

- The BlackBerry MDS software requires the 32-bit JDK to be installed, even for 64-bit operating systems.
- A registered BlackBerry developer account to download the JDE.

Task

Go to the BlackBerry Web site to download and install the BlackBerry JDE.
The MDS-CS simulator is installed with the BlackBerry JDE.

Installing X.509 Certificates on BlackBerry Devices and Simulators

Install the .p12 certificate on the BlackBerry device or simulator and select it during authentication. A certificate provides an additional level of secure access to an application, and may be required by an organization's security policy.

1. Install the certificate on a device.
   a) Connect to the device with a USB cable.
   b) Browse to the SD Card folder on the computer to which the device is connected.
   c) Navigate to and select the certificate. Enter the password.
   d) Import the certificate.
2. Install the certificate on a simulator.
   a) From the simulator, select **Simulate** > **Change SD Card**.
   b) Add/or select the directory that contains the certificate.
   c) Open the media application on the device, and select **Menu** > **Application** > **Files** > **MyFile** > **MediaCard**.
   d) Navigate to and select the certificate. Enter the password.
   e) Check the certificate and select **Menu** > **Import Certificate**. Click **Import Certificate** then enter the data vault password.

Creating Projects and Adding Libraries into the BlackBerry Development Environment

Set up the BlackBerry project and add required libraries. Use these procedures if you are developing a device application using the BlackBerry JDE or the BlackBerry Java plug-in for Eclipse.

Adding Required .jar Files

Add the following Online Data Proxy .jar file references to the BlackBerry project's Java build path.

Copy the following SDM .jar files:

- `sdmcache-2.3.0-preverified.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `sdmcommon-2.3.0-preverified.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `sdmconfiguration-2.3.0-preverified.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `sdmconnectivity-2.3.0-preverified.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `sdmparser-2.3.0- preverified.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `sdmpersistence-2.3.0-preverified.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `sdmsupportability-2.3.0-preverified.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.

Copy the following ODP .jar files:

- `MCL.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `sup_json.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `SUPProxyClient-2.3.0.jar` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.
- `bb-perflib-1.6` – from *SMP_HOME*`\MobileSDK\OData\BB\libraries\` for the BlackBerry client.

Consuming Java .JAR files for BlackBerry Projects

Add the .jar d files to your BlackBerry project.

Using this procedure, the Java definitions are available in Eclipse in order to find the third-party classes when compiling your project's source code. After compilation you will have one `.cod` file containing the application and the libraries together.

1. Download the library to your host development system.
2. Create a new folder, named `libs`, in your Eclipse/BlackBerry project.
3. Right click `libs` and choose **Import -> General -> File System**, then click **Next**.

4. Browse the file system to find the library's parent directory (where you downloaded it).
5. Click **OK**, then click the directory name (not the checkbox) in the left pane and check the relevant JAR in the right pane. This puts the library into your project (physically).
6. Right click on your project, choose **Build Path -> Configure Build Path**, then click the **Libraries** tab, then click **Add JARs...**
7. Navigate to your new JAR in the libs directory and add it.
8. Click on the **Order** and **Export** tab. After you added the libraries they should be listed. Check all the libraries. This way the libraries will be compiled together with the application and packaged into one `.cod` file.

Development Task Flow for BlackBerry Applications

For an application to work in an ODP scenario, it needs to be initialized first. Additionally, an application developer uses APIs specific to ODP that enables the application to send and receive data.

1. *Initializing the Application*

   Initialize the application when it starts the first time.
2. *Setting Server Details*

   Set or update the connection properties of the server before user registration.
3. *Registering a User*

   Using a pre-defined authentication mechanism, register a user automatically. You can register the user synchronously or asynchronously.
4. *Sending Data Request to the Back-end*

   You can forward data request messages to the back-end through theSUP server synchrounously or asynchronously.
5. *Retrieving the Response from the Back-end*

   Based on the data request sent to the back-end, you need to retrieve the data as a response to the device.
6. *Using HTTPS over the SAP Mobile PlatformMessaging Channel*

   (Optional) Clients can securely communicate with the backend via the server through the HTTPS transport protocol. By using SSL over the SAP Mobile Platform Messaging Channel, all messages are encrypted and authenticated to ensure secure communication.
7. *Debugging Runtime Error and Performance Analysis*

   To handle occurrences of exceptions and special conditions that change the normal flow of the program execution, error handling has to be done appropriately.

See also

- *Getting Started Task Flow for BlackBerry Applications* on page 313

- *Deploying Applications to Devices* on page 327

Initializing the Application

Initialize the application when it starts the first time.

The following illustrates how to initialize an application.

```
ODPUserManager.initInstance("Testing");
ODPUserManager userApp = ODPUserManager.getInstance();
```

See also

- *Setting Server Details* on page 319

Registering Listeners for Push Notifications

(Optional) The application should register and implement a listener interface to receive native or payload push notifications.

The `IODPPushNotificationListener` interface should be implemented by the application to receive the BES native push notifications on BlackBerry devices. The `onPushNotification` method is called when a new BES push notification is received. For more information on configuring BES native notifications on SAP Control Center, see *BES Native Notification Properties* in *SAP Control Center for SAP Mobile Platform.*

Note: When the application is not running or terminated, the BES notifications will not reach the client device. The IODPPushNotificationListener will not be invoked.

The following code illustrates how to register listeners for native push notifications.

```
Public class Mylistener implements IODPPushNotifciationListner
{
    Public int onPushNotifcation(Hashtable ht)
    {
          If(getpayload == true)
             Return 0;
          Else
             Return 1;
    }
}
```

The return values for onPushNotification include:

- 0 – The client receives the payload notification from the server, if the **online/payload push with native notification** option is selected in the **Push Configurations** tab. For more information, see *Configuring Native Notifications* in *SAP Control Center for SAP Mobile Platform.*
- 1 – The client does not receive the payload notification from the server.

Enabling Online Push

To consume push messages, the application registers a listener object.The client SDK notifies this listener object whenever there is a push message from the server. The listener object should implement the ISDMNetListener interface.

Syntax

```
public static void
setPushListener(com.sap.mobile.lib.sdmconnectivity.ISDBNetListener
pushListerner)
```

Parameters

- **pushListener –** Object that implements ISDMNetListener interface.

Examples

- **Listener Object**

  ```
  UserManager.setPushListener(listenerObjectFromApp);
  ```
- **Implementation of APIs in the Listener Object**

  ```
  ISDMNetListener.onError(ISDMRequest, IHttpResponse,
  ISDMRequestStateElement)
  ISDMNetListener.onSuccess(ISDMRequest, IHttpResponse,
  ISDMRequestStateElement)
  ```

Setting Server Details

Set or update the connection properties of the server before user registration.

The following code illustrates how to set the server details.

```
userApp.setConnectionProfile("155.56.51.245", 123, "test.farmMBS");
```

See also

- *Initializing the Application* on page 318

Registering a User

Using a pre-defined authentication mechanism, register a user automatically. You can register the user synchronously or asynchronously.

The following code illustrates how to register a user automatically.

```
userApp.registerUser("perfios" , "SSO" , "perfios" , true);
```

Sending Data Request to the Back-end

You can forward data request messages to the back-end through theSUP server synchrounously or asynchronously.

1. *Creating a URL Request*

   Create a URL request that enables the device to forward a data request to the corresponding back-end.
2. *Enabling XSRF Token Protection*

   For all modifying operations such as POST, PUT and DELETE, the XSRF token is used in the HTTP request-response header field.
3. *Assigning Credentials*

   Assign the user credentials.
4. *Adding Custom Headers*

   Add custom headers to a request message. This is a name/value pair that defines the operating parameters of an HTTP transaction. Custom headers are optional while sending a data request to the back-end.
5. *Setting the Required Timeout*

   Set the timeout value upto which the application waits until the request reaches the server.
6. *Forwarding the Request*

   Send the asynchronous request message to the back-end.

See also

- *Retrieving the Response from the Back-end* on page 322

Creating a URL Request

Create a URL request that enables the device to forward a data request to the corresponding back-end.

The following code illustrates how to create a URL request.

```
ISDMConnectivitiyParameters parameters = new
SDMConnectivityParameters();
parameters.setUserName("XYZ");
parameters.setUserPassword("ABC");
//Where XYZ = backend username and ABC = backend password.
parameters.setBaseUrl("<server-address>/sap/opu/sdata/sap/
FINCUSTFACTSHEET/");
```

Enabling XSRF Token Protection

For all modifying operations such as POST, PUT and DELETE, the XSRF token is used in the HTTP request-response header field.

Once you have enabled XSRF token on the device, the token is extracted after the first GET request triggered by the application. This token is retained in the device memory, and is added to all subsequent modifying requests.

The following code illustrates how to enable XSRF token.

```
parameter.enableXSRF(true);
```

Assigning Credentials

Assign the user credentials.

The following code illustrates how to assign credentials.

```
parameters.setUserName("smpuser123");
parameters.setUserPassword("456");
parameteres.setLanguage("en");

ISDMPreferences preferences = new SDMPreferences();
preferences.setPreference(ISDMPreferences.SDM_CONNECTIVITY_HANDLER_
CLASS_NAME,com.sybase.mobile.lib.client.SUPConnectionFactory.class.
getName());
preferences.setPreference(ISDMPreferences.SAP_APPLICATIONID_HEADER_
VALUE,"MyApp.2.3.0.0");

ISDMLogger logger = new SDMLogger(preferences);
ISDMRequestManager RequestManager = new SDMRequestManager(logger,
preferences, parameters, 1);

ISDMRequest serviceDocumentRequest = new SDMNamedRequest(1);
serviceDocumentRequest.setPriority(ISDMRequest.PRIORITY_HIGH);
serviceDocumentRequest.setRequestMethod(ISDMRequest.REQUEST_METHOD_
GET);
serviceDocumentRequest.setRequestUrl("<server-address>/sap/opu/
sdata/sap/FINCUSTFACTSHEET/?sap-
language=en;ConnectionTimeout=50000");
```

Adding Custom Headers

Add custom headers to a request message. This is a name/value pair that defines the operating parameters of an HTTP transaction. Custom headers are optional while sending a data request to the back-end.

The following code illustrates how to add custom headers.

```
Hashtable headers = new Hashtable();
headers.put("X-Requested-With", "XMLHttpRequest");
serviceDocumentRequest.setHeaders(headers);
```

Setting the Required Timeout

Set the timeout value upto which the application waits until the request reaches the server.

The following code illustrates how to set the required timeout.

```
preference.setPreference(ISDMPreferences.CONNECTION_TIMEOUT_MS,
"1000");
```

Forwarding the Request

Send the asynchronous request message to the back-end.

The following code illustrates how to forward an asynchronous message.

```
serviceDocumentRequest.setListener(this);
RequestManager.makeRequest(request);
```

Retrieving the Response from the Back-end

Based on the data request sent to the back-end, you need to retrieve the data as a response to the device.

The following code illustrates how to retrieve the response from the back-end.

```
public void onSuccess(ISDMRequest aRequest, IHttpResponse aResponse)
{
String serviceDocument = new String(aResponse.getContent());
int resCode = aResponse.getStatusCode();
UiApplication.getUiApplication().invokeLater(new Runnable()
{
public void run()
{
Dialog.inform("success"+ resCode);
Dialog.inform("Content"+ serviceDocument);
}
});
}
```

See also

- *Sending Data Request to the Back-end* on page 320

Using HTTPS over the SAP Mobile PlatformMessaging Channel

(Optional) Clients can securely communicate with the backend via the server through the HTTPS transport protocol. By using SSL over the SAP Mobile Platform Messaging Channel, all messages are encrypted and authenticated to ensure secure communication.

See also

- *Debugging Runtime Error and Performance Analysis* on page 325

Enabling HTTPS as a Transport Protocol

You can set HTTPS as the transport protocol that the OData client should use to communicate with any host (Example: Relay server).

Syntax

```
public void enableHTTPS(boolean useHTTPS) throws ODPException
```

Parameters

- **useHTTPS –** Set to "true", if the protocol to be used is HTTPS.

  Set to "false", if the protocol to be used is HTTP.

  Note: The default protocol is HTTP.

Verifying a Server Certificate

The underlying platform for BlackBerry devices does not enable the application to govern the server verification process.

If the server certificate is trusted, the connection is successful. If the server certificate is not trusted, the OS pops a message that requires the user to manually accept the certificate. If the certificate is accepted, the connection is established else the certificate is rejected and the connection fails.

Note: HTTPS on BlackBerry works considerably slower than HTTP. The use of HTTPS acts as a overhead since secure communication is ensured through BES by default.

Enabling a Listener for HTTPS Support with Basic Authentication Challenge

When a client attempts to connect to a host (like a relay server), this connection is authenticated with a basic authentication challenge.To setup a HTTP basic authentication, the application registers a listener with the OData SDK. If the IODPHTTPAuthChallengeListener is not registered, an HTTP_AUTH_FAILURE error is returned when a challenge is received.

Syntax

```
public static void
setODPHTTPAuthChallengeListener(ODPClientListeners.IODPHTTPAuthChal
lengeListener listener) throws ODPException;
```

Parameters

- **listener –** Listener object that implements the ODPClientListeners.IODPHTTPAuthChallengeListener.The object invokes the callback method of this interface when a connection receives the HTTP_AUTH_FAILURE (HTTP response code 401) challenge

Examples

- **Register a listener for HTTP(s) Auth Challenge –**

  ```
  ODPUserManager.enableHTTPS(true);
  ODPUserManager.setODPHTTPAuthChallengeListener(this);
  ```

- **Implementation of the Listener**

  ```
  public class UserRegistration implements
  IODPHTTPAuthChallengeListener
  {
        .
        .
       public void startUserRegistration(){
           ODPUserManager.initialize(appID);

  ODPUserManager.setConnectionProfile(serverIP,serverPort,farmID);
           ODPUserManager.enableHTTPS(true);
           ODPUserManager.setODPHTTPAuthChallengeListener(this);

  ODPUserManager.registerUser(username,activationCode,false);

  }

  // callback method for HTTP authentication 401 challenge
  public ODPHTTPAuthChallengeCredentials getCredentials(String
  sHostName,String oldUserName, String realm) {
  // query the user for credentials, username and password for
  sHostName
          .
          .
          .
  // return the credentials in ODPHTTPAuthChallengeCredentials
  structure.

              .
  return new ODPHTTPAuthChallengeCredentials(username,password);



          }
  ```

Registering a Listener for HTTP Error in BlackBerry Applications

To ensure that OData BlackBerry clients are notified of HTTP errors while establishing a connection with the network edge, implement a listener.

Syntax

```
public static void setODPHTTPErrorListener(IODPHttpErrorListener
oListener) throws MessagingClientException
```

Parameters

- **oListener –** listener object that implements the interface `IODPHttpErrorListener`.

Examples

- **Implement the listener**

```
public class UserRegistration implements IODPHttpErrorListener{
 .
 .
   public void startUserRegistration(){
     UserManager.initialize(appID);
     UserManager.setConnectionProfile(serverIP,serverPort,
farmID);
     UserManager.setODPHttpErrorListener(this);
     UserManager.registerUser(username,securityConfig,password);
     }

  //callback method for HttpError
  public void onHttpError(int errorCode, String errorMsg,
Hashtable errorHeader) {
    logger.info(null, "On HttpError", "Error Info" +errorCode
+errorMsg);
  }
}
```

Debugging Runtime Error and Performance Analysis

To handle occurrences of exceptions and special conditions that change the normal flow of the program execution, error handling has to be done appropriately.

The following code illustrates how to handle errors for asynchronous requests.

```
public void onError(ISDMRequest aRequest, IHttpResponse aResponse,
ISDMRequestStateElement aRequestStateElement))
{
int errCode = aRequestStateElement.getErrorCode();
String str = new String(aResponse.getContent());
serviceDocument = "Error Code: "+ errCode + str;
}
```

See also

- *Using HTTPS over the SAP Mobile PlatformMessaging Channel* on page 322

Analyzing Performance Data Points

(Optional) To analyze the performance of the client, measurement points are available at different stages in a request-response cycle. These points are used to provide logs that help in assessing the processing time across various components in the SAP Mobile Platform environment.

Note: The response time will be impacted after importing performance library. If you set the custom settings in SAP Control Center as true for logs, then there will be a degradation in the response time.

Data Points for the Client

The table below provides the list of data points and the log readings across which the performance can be measured.

Log Reading	**Application**	**Proxy Client**	**Messaging Client**	**Network (includes reverse proxy/ relay server)**	**SAP Mobile Platform Server**	**Network**	**Enterprise Information System (EIS)**
E2E: RR		X	X	X	X	X	X
ODP:RR			X	X	X	X	X
IMO:RR			X	X	X	X	X
Net-work:RR				X	X	X	X

Performance Readings

The above log readings determine the time elapsed across various stages of a request response (RR) cycle.

- E2E:RR - Corresponds to the performance reading when the request is forwarded from the proxy client and the response reaches the proxy client.
- ODP:RR - Corresponds to the performance reading when the request reaches the Messaging Client and the response reaches the Messaging Client.
- IMO:RR - Corresponds to the performance reading when the request is forwarded from the Messaging Client and the response reaches the Messaging Client.
- Network:RR - Corresponds to the performance reading when the request is forwarded from the network and the response reaches the network.

The above log readings determine the time elapsed across various stages of a request response (RR) cycle. Use these readings to determine processing time across the following components:

- Time taken at the messaging client: **ODP:RR** - **IMO:RR**
- Time taken at the proxy client: **E2E:RR** - **ODP:RR**

Deploying Applications to Devices

This section describes how to deploy customized mobile applications to devices.

1. *Signing*

   Code signing is required for applications to run on physical devices.
2. *Provisioning Options for BlackBerry Devices*

   To provision the application to BlackBerry devices, you can automatically push the application to the device or send a link to device users so they can install it when desired. For small deployments or evaluation purposes, device users can install the application using BlackBerry Desktop Manager.
3. *BES Provisioning for BlackBerry*

   BlackBerry devices that are connected to a production environment using relay server can use BlackBerry Enterprise Server (BES) to provision supported device types.
4. *BlackBerry Desktop Manager Provisioning*

   You can deploy BlackBerry applications to physical devices through BlackBerry Desktop Manager.

See also

- *Development Task Flow for BlackBerry Applications* on page 317
- *OData SDK Components and APIs* on page 329

Signing

Code signing is required for applications to run on physical devices.

In general, if your application or library uses an API it must be signed, which occurs in most cases. The executables are generated when the application project is signed.

You can implement code signing from the BlackBerry JDE:

- BlackBerry JDE – download the Signing Authority Tool from the BlackBerry Web site at *http://na.blackberry.com/eng/developers/javaappdev*. View Deploying and Signing Applications in the BlackBerry JDE plug-in for Eclipse at the Research In Motion Developer Video Library Web site: *http://supportforums.blackberry.com/t5/Java-Development/tkb-p/java_dev%40tkb?labels=video*.

Provisioning Options for BlackBerry Devices

To provision the application to BlackBerry devices, you can automatically push the application to the device or send a link to device users so they can install it when desired. For small deployments or evaluation purposes, device users can install the application using BlackBerry Desktop Manager.

Once installed on the device, the application appears in Downloads. However, device users can move it to a different location. If device users reinstall the application from a link or URL, or using Desktop Manager, the BlackBerry device remembers the installation location.

Provisioning Method	Purpose	Description
BlackBerry Enterprise Server (BES) Over-the-Air (OTA)	Enterprise installations	When the BlackBerry device activates, it automatically pairs with the BES and downloads the application. See *http://www.blackberry.com/btsc/search.do?cmd=displayKC&docType=kc&externalId=KB03748* for step-by-step instructions.
OTA: URL/link to installation files	Enterprise installations	The administrator stages the OTA files in a Web-accessible location and notifies BlackBerry device users via an e-mail message with a link to the JAD file.
Desktop Manager	Personal installation	Installs the application when the BlackBerry device is synced via a computer.

BES Provisioning for BlackBerry

BlackBerry devices that are connected to a production environment using relay server can use BlackBerry Enterprise Server (BES) to provision supported device types.

See *Provisioning Options for BlackBerry Devices* in *Mobile Application Provisioning* for details on how to perform BlackBerry provisioning.

BlackBerry Desktop Manager Provisioning

You can deploy BlackBerry applications to physical devices through BlackBerry Desktop Manager.

The application code is compiled against the BlackBerry RAPC compiler to output the following COD (.cod), Application Loader Files (.alx), and Java Application Descriptor (.jad) files. File requirements depend on application and installation type.

OData SDK Components and APIs

The OData SDK for BlackBerry provides the means to easily build an application which relies on the OData protocol and its additions made by SAP.

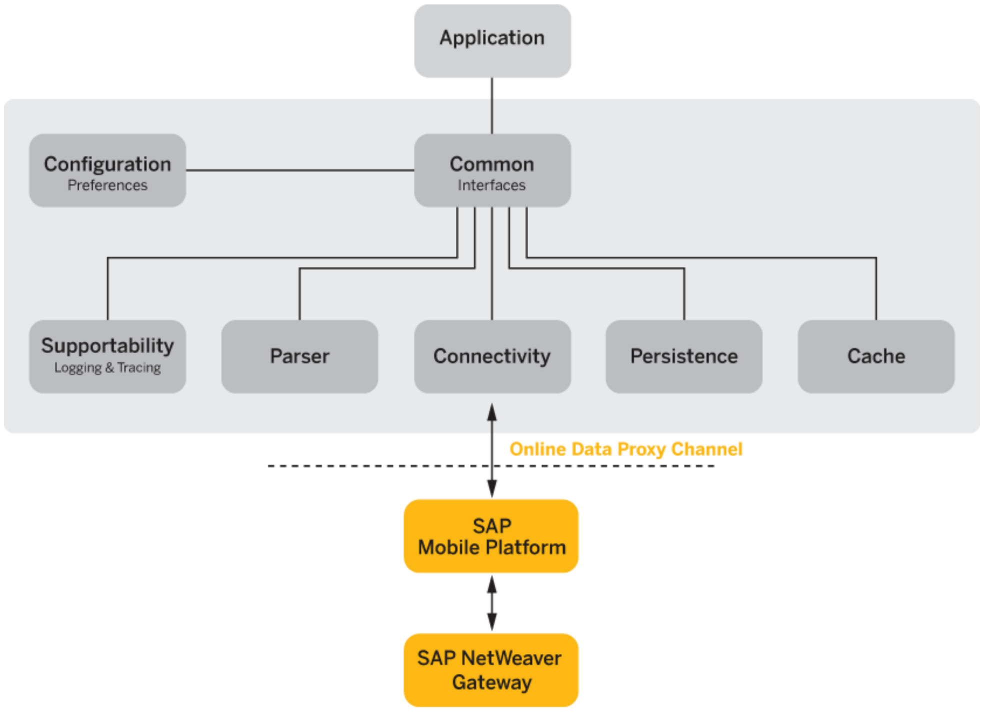
Prerequisites

Download the Eclipse IDE and the BlackBerry java plug-in for Eclipse to be able to develop on BlackBerry platform.

OData SDK - BlackBerry

The full list of APIs and their descriptions are available after the installation of SAP Mobile Platform at the following location within your installation folder: `...\UnwiredPlatform\MobileSDK\OData\BlackBerry\docs`.

The following figure shows the main components of the OData SDK on BlackBerry.



SDMCommon

To build an application on the OData SDK, you must first import the SDMCommon component that contains interfaces and configuration for the components. None of the components have dependency on each other, but all of them depend on the SDMCommon component, and all of them have references to interfaces of other components (held by SDMCommon).

Component Replacements

In your own application, you can replace the implementation behind an interface of a BlackBerry OData SDK component. For example, if you want to add a new functionality to SDMCache, but keep everything else unchanged (for example, the way it is persisted by SDMPersistence) you can implement your own solution. The new cache can be either a new implementation, or a descendant of SDMCache, as long as it implements the ISDMCache interface from SDMCommon.

See also

- *Deploying Applications to Devices* on page 327
- *ODP SDK API Usage* on page 349

SDMParser

The parser (SDMParser class) is the core component of the package, it is responsible for processing XMLs. The actual parsing is done by the standard java SAX parser provided by the BlackBerry platform.

Parsing is generic in the sense that an arbitrary (well-formed) XML can be processed, and the information content is returned without any loss:

```
/**
 * Parses the stream source of an XML and converts it to a Java Object
containing all
 * the information that were contained by the source XML.
 *
 * @param xml
 *            A byte array that holds a syntactically valid XML.
 * @return ISDMParserDocument The Object representation of the parsed
XML.
 * @throws SDMParserException
 *             If the XML source is invalid.
 * @throws IllegalArgumentException
 *             If the argument is null.
 */
public abstract ISDMParserDocument parseXML(byte[] xml) throws
SDMParserException, IllegalArgumentException;
```

The ISDMParserDocument interface provides access to all the data stored in the XML. The API user constructs the path inside the XML to the given data (attribute or text value), then the following methods return their value:

```
/**
* Returns the string value of the sub-document contained by this
object and accessible via the
* element names provided by the 'route' argument.
*
* @param route
*            "/" separated route that leads to the object route must
contain indexes as well.
*             Route must end with the index number which uniquely
identifies an XML element.
*             route must start with "/"
* @return The string value of the XML element on the route. It
returns null if route
*          does not identify a unique element.
*/
public abstract String getValue(String route);

/**
* Returns the string value of the XML attribute of the object
accessible via the element names
* provided by the 'route' argument.
*
* @param route
*            "/" separated route that leads to the object route must
contain indexes as well.
*             Route must end with the index number which uniquely
identifies an XML element and
*            after the element the attribute local name (field name)
must be appended with
*           slash. Example route: "/element1/1/element2/5/element3/2/
attributename"
* @param namespaceURI
*           The Namespace URI of the attribute, or the empty String if
the attribute local
*             name has no Namespace URI.
*
* @return The string value of the given attribute on the given route
*/
public abstract String getAttribute(String route, String
namespaceURI) throws IllegalArgumentException;
```

However, for applications that communicate with the OData Protocol and that are working with OData objects, it is more suitable to use parser methods that provide OData objects (hierarchies).

There are specific parser methods for the document types that come in the OData Protocol responses. These are the service document, metadata document, open search description, error message, atom feed and entry:

```
/**
* Parses the SDMOData Service Document XML and converts it to an
appropriate Java Object.
*
* @param xml
*             The byte array that holds SDMOData Service Document XML
* @return ISDMODataServiceDocument The Object representation of
SDMOData Service Document.
* @throws SDMParserException
*              If the XML source is invalid.
* @throws IllegalArgumentException
*              If the argument is null.
```

```
*/
public abstract ISDMODataServiceDocument
parseSDMODataServiceDocumentXML(byte[] xml) throws
SDMParserException, IllegalArgumentException;

/**
* Parses the SDMOData metadata XML and converts it to an appropriate
Java Object.
*
* @param xml
*            The byte array that holds SDMOData Schema XML
* @return ISDMODataSchema The Object representation of the SDMOData
Schema.
* @throws SDMParserException
*             If the XML source is invalid.
* @throws IllegalArgumentException
*             If the argument is null.
*/
public abstract ISDMODataMetadata parseSDMODataMetadataXML(byte[]
xml, ISDMODataServiceDocument svDoc) throws SDMParserException,
IllegalArgumentException;
```

The service document XML has to be processed before the metadata, because metadata parsing needs the service document object.

```
/**
* Parses the SDMOData Open Search Description XML from stream and
converts it to an *appropriate Java Object.
*
* @param xml
*            The byte array that holds the SDMOData Open Search
Description XML.
* @return ISDMODataOpenSearchDescription The Object representation
of the SDMOData Open Search
*         Description.
* @throws SDMParserException
*             If the XML source is invalid.
* @throws IllegalArgumentException
*             If the argument is null.
*/
public abstract ISDMODataOpenSearchDescription
parseSDMODataOpenSearchDescriptionXML(byte[] xml) throws
SDMParserException, IllegalArgumentException;

/**
* Parses the SDMOData Error XML from stream and converts it to an
appropriate Java Object.
*
* @param xml
*            The byte array that holds the SDMOData Error XML.
* @return ISDMODataError The Object representation of the SDMOData
Error.
* @throws SDMParserException
*             If the XML source is invalid.
* @throws IllegalArgumentException
*             If the argument is null.
```

```
*/
public abstract ISDMODataError parseSDMODataErrorXML(byte[] xml)
throws SDMParserException;
```

There are also dedicated methods for feed and parsing, and the parser is also able to process entry XMLs. Both of them need the entity set object representing the collection container of the entry, so the parser has access to the metadata of the entry, which is needed for proper data parsing.

```
/**
* Parses OData XML structures from stream that represent either a
single SDMOData Entry or a
* feed of several SDMOData entries.
*
* @param xml
*            The byte array that holds the XML source of either a
single SDMOData Entry or a
*            feed of several SDMOData entries.
* @return ISDMODataFeed The vector of the SDMOData Entries contained
by the source XML.
* @throws SDMParserException
*             If the XML source is invalid.
* @throws IllegalArgumentException
*             If the argument is null.
*/
public abstract ISDMODataFeed parseSDMODataEntriesXML(byte[] xml,
ISDMODataEntitySet eSet)
throws SDMParserException, IllegalArgumentException;

/**
* Parses an entry XML.
*
* @param xml
*            byte array the data is read from
* @param eSet
*            the related entity type
* @return ISDMODataEntry
* @throws SDMParserException
* @throws IllegalArgumentException
*/
public ISDMODataEntry parseSDMODataEntryXML(byte[] xml,
ISDMODataEntitySet eSet) throws SDMParserException,
IllegalArgumentException;

/**
      * This method is called if performance data point are to be
enabled for Parser
      * @param perflog
      *  a boolean value to enable or disable performance logs
      */
      public void enableParserPerformanceLog(boolean perflog);
```

All the OData related classes are descendants of the generic `SDMParserDocument` class, meaning that its low level data access methods can be applied for the OData classes as well.

This feature is useful when some information from the XML files is not accessible through the high level interfaces.

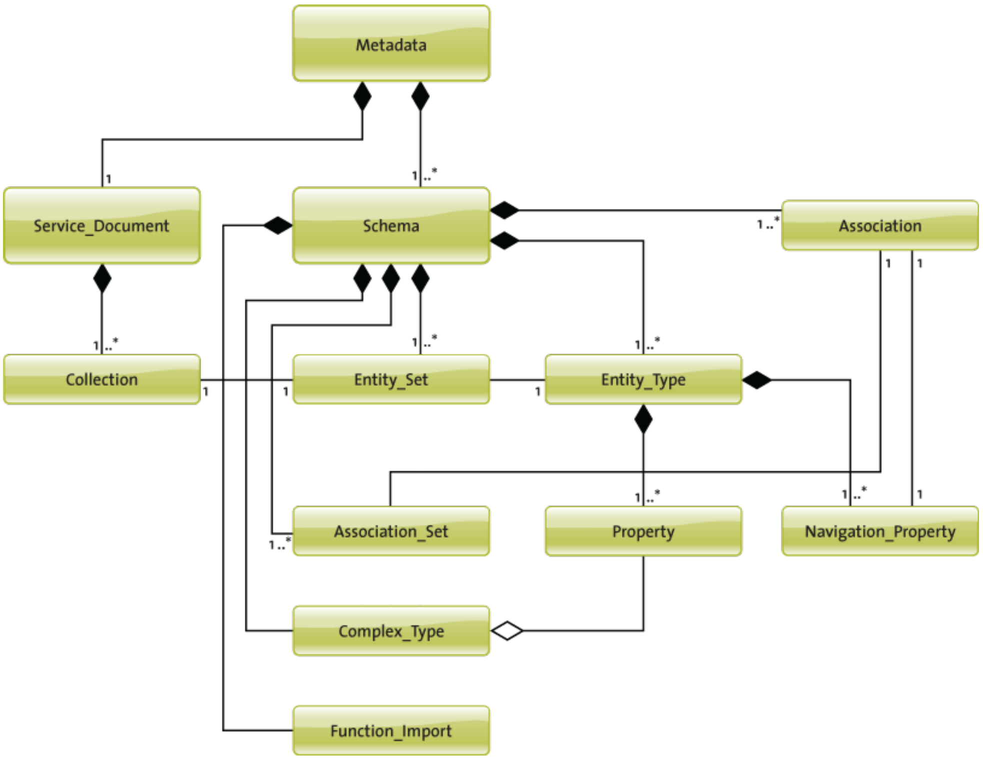
The structure of the metadata classes is built according to the OData object hierarchy. The information is accessed from two XMLs, the service document and the metadata XML. The service document is parsed first, then the metadata. The ISDMODataMetada object, which is received from the parser after processing the metadata XML is the root of the hierarchy. From this starting point, you can browse the whole hierarchy. Furthermore, from each lower level object, you can access its parent using the public `ISDMParserDocument getParent()` method. The ISDMParserDocument is the parent of all OData classes, so the result can be type cast to the proper OData type.

Collections and entity sets are in one-to-one relationship, containing even partially overlapping meta information about the corresponding atom feeds. This relationship is implemented through their name attribute, their non-qualified name is the same. However, used as method parameters, collection name is always without namespace, while entity set name is prefixed with the corresponding schema namespace.

Parsing is done without any data loss, that is, all the information contained in the XML is preserved in the resulted data structures. In addition to these data structures, the complete XML is also preserved. This is useful when the objects are persisted, because it is more efficient to persist a simple string instead of a complex data structure. It is also an advantage when data is stored encrypted.

The only drawback of this solution is when data is restored from the persistent storage, the stored XMLs are parsed again. So this is an expensive operation and should be done as rarely as possible. To avoid degrading user experience, application developers should perform this operation (restore object structure from persistence) in a background thread.

Figure 1: Object Hierarchy



There are certain use cases, where OData entry objects and their XML representations have to be created on the client side. For this, the `SDMDataEntry` class provides the public constructor `SDMDataEntry(ISDMODataEntitySet eSet)`, which creates an empty entry object so that its attributes have to be set one-by-one by calling the corresponding setter method. Finally, the public `String toXMLString()` method generates the XML representation of the entry object.

ETag Support

An entity tag is one of the several mechanisms that HTTP provides for cache validation and which allows a client to make conditional requests. An ETag is an identifier assigned by a Web Server to a specific version of a resource found at a URL. If the resource content at the URL changes, a new and different ETag is assigned.

Examples

Provides example snippets to understand the use of SDMParser APIs

ETag Support

```
private SDMODataEntry entryDoc;  //entryDoc corresponds to the XML
document with entries
```

```
String eTag = entryDoc.getEtag();  //Passes a null value if the
document does not contain an ETag.
```

SDMCache

The SDMCache component is responsible for storing and accessing OData related objects in the memory of the device.

List of Features

- Storing ISDMODataEntry objects in the memory
- Accessing ISDMODataEntry objects in the memory directly by their key
- Searching for ISDMODataEntry objects in the memory using tokenized prefix search on their searchable fields
- Managing the number of stored ISDMODataEntry objects based on the maximum size of the capacity, removing the least recently used OData document first

SDMCache Public APIs

Provides a list of public APIs in the SDMCache library.

SDMCache Public APIs

```
ISDMCache

 void initialize(ISDMPreferences preferences);
 void clear();
 void setSDMODataServiceDocument(ISDMODataServiceDocument
serviceDocument);
  void setSDMODataMetadata(ISDMODataMetadata metadata)
 void setSDMODataEntry(ISDMODataEntry entry, String collectionId);
 void setSDMODataEntries(Vector entries, String collectionId);
 ISDMODataServiceDocument getSDMODataServiceDocument();
 ISDMODataSchema getSDMODataSchema();
 ISDMODataMetadata getSDMODataMetadata()
 ISDMODataEntry getSDMODataEntry(String key);
 Vector getSDMODataEntries(String collectionId);
 Vector getStoredDocuments();
 Vector searchSDMODataEntries(String searchTerm, String
collectionId);
 void removeSDMODataServiceDocument();
 void removeSDMODataSchema();
 void removeSDMODataEntry(String key);
 void removeSDMODataEntries(String collectionId);
 void removeStoredDocuments();
 Hashtable getStoreStructureForPersistency();
 void setStoreStructureForPersistency(Hashtable values);
```

Technical Details

For capacity management, SDMCache uses an LRU (least recently used) algorithm that ensures that the least recently used entries are removed first because of reaching the maximum capacity. Maximum number of capacity can be set using preference with key:

ISDMPreferences.CACHE_MAX_ELEMENT_NR. This setting refers to the maximum number of cached entities per Collection.

SDMCache supports the tokenized prefix search. The gp:use-in-search property tag determines whether a field is searchable.

SDMCache depends on OData specific interfaces of SDMParser, but does not depend on the real implementation of SDMParser.

SDMPersistence

The Persistence layer stores the application's state and relevant data on the mobile device using the BlackBerry Persistent Store. The library exposes secure APIs, allowing encrypted data storage and decryption of data.

List of Features

- Storing and loading general objects from Persistent store
- Storing and loading the SDMCache object
- Storing and loading the SDMCache object in a secured way, which means that all fields of all objects within the cache will be encrypted/decrypted during the load/store operations. There is a specific method for the removal of the cache, but for the general objects, just a generic method is provided, where the persistent object id has to be provided as parameter.

SDMPersistence Public APIs

Provides a list of public APIs in the SDMPersistence library.

SDMPersistence Public APIs

```
ISDMPersistence

void storeCache(final ISDMCache cache)
void storeCacheSecured(final ISDMCache cache)
void storePreferencesSecured(final ISDMPreferences preferences)
ISDMCache loadCache(ISDMCache cache, ISDMParser parser)
ISDMCache loadCacheSecured(ISDMCache cache, ISDMParser parser)
void loadPreferencesSecured(final ISDMPreferences preferences)
void storeObject(final long key, final Object object)
Object loadObject(final long key)
void clearCache()
void clearObject(final long key)
```

Technical Details

To persist data on the BlackBerry platform means storing objects in the storage provided by the platform (Persistent Store). Data is stored as instances of Persistent Objects. A PersistentObject can be any object that implements the Persistable interface. The Persistent Store API allows the implicit persistence of classes, so the following data types automatically implement the Persistable interface and can also be stored in the persistent store:

- `java.lang.Boolean`
- `java.lang.Byte`
- `java.lang.Character`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Object`
- `java.lang.Short`
- `java.lang.String`
- `java.util.Vector`
- `java.util.Hashtable`

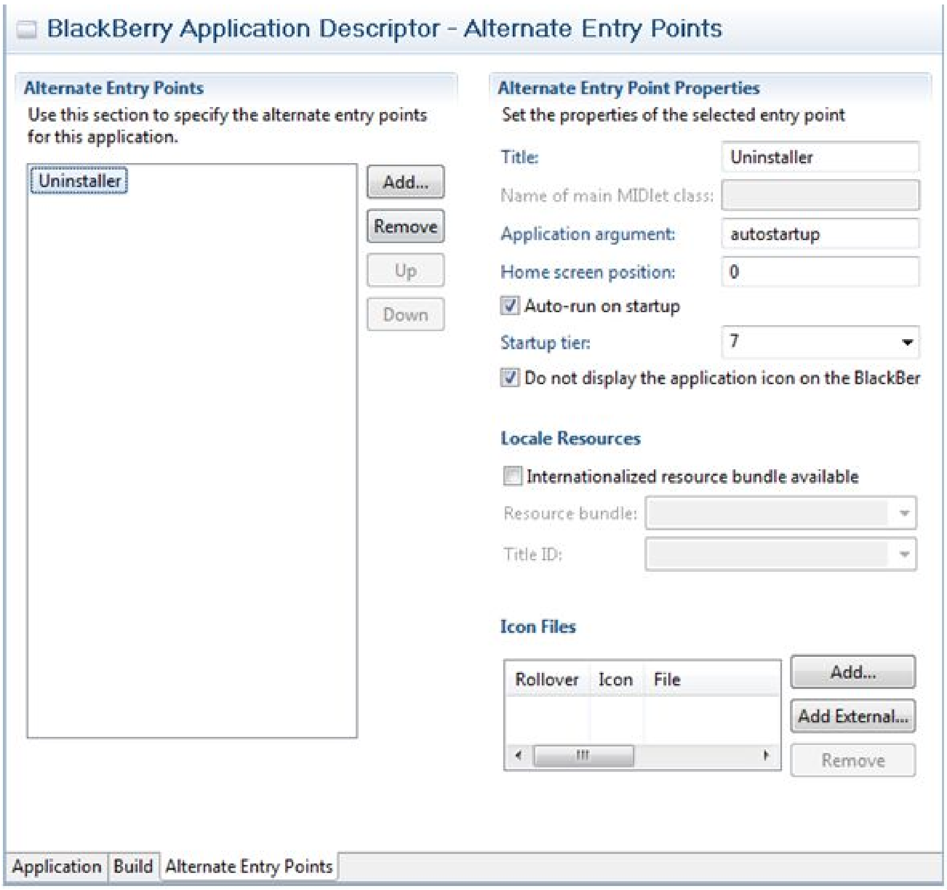
The implementation only uses the above standard data types when persisting data. The persistent class cannot be used by two applications on the same device of the BlackBerry platform, and hence is not suitable for a static library component. In addition, this also avoids any limits on the number of custom persistent classes supported by the platform.

The storage for each application is distinct, because each object in the persistent store is associated with a 64-bit ID (type long). Data is stored in the Persistent Store which is a fast and optimized storage on the platform. The BlackBerry Persistent Store APIs are designed to provide a flexible and robust data storage interface. With the BlackBerry Persistent Store APIs, you can save entire Java objects to the memory without having to serialize the data first. When the application is started, it can retrieve the Java object from the memory and process the information. No size limit exists on a persistent store; however, the limit for an individual object within the store is 64 KB.

When using standard persistent classes, each application must ensure to remove any persisted objects when the application is removed from the device. The BlackBerry OS does not automatically remove these objects in the same way as it does for custom persistent classes.

The applications have to implement the `CodeModuleListener` interface, which can react to module addition and removal events. Register the implementation to the `CodeModuleManager` with the `public static void addListener(Application application, CodeModuleListener listener)` method. The first parameter is the application whose event listener thread will execute the listener's code. This means that this application process must be running when the application removal is triggered. This can be achieved by adding an automatically starting background process to the applications and register the listener there.

An alternate entry point with automatic startup has to be added to the application descriptor:


BlackBerry Application Descriptor - Alternate Entry Points

Alternate Entry Points

Use this section to specify the alternate entry points for this application.

Uninstaller

Alternate Entry Point Properties

Set the properties of the selected entry point

Title:	Uninstaller
Name of main MIDlet class:	
Application argument:	autostartup
Home screen position:	0
☑ Auto-run on startup	
Startup tier:	7
☑ Do not display the application icon on the BlackBer	

Locale Resources

☐ Internationalized resource bundle available

Resource bundle:

Title ID:

Icon Files

The main method of the application has to be extended with a branch for the background process, which registers itself for code module changes:

```
public static void main(String[] args) {
  if (args.length >= 1 && args[0].equals("autostartup")) {

  // Background startup of the application. This process registers as
the listener for
  // code module life-cycle changes. This will be an always on
background process, which
  // will react, when its own module is marked for deletion.
    UninstallSampleApp theApp = new UninstallSampleApp(false);
    CodeModuleManager.addListener(theApp, theApp);

    theApp.requestBackground();
    theApp.enterEventDispatcher();
  } else {
  // Normal startup procedure: create a new instance of the
application which will run in
  // the foreground.
```

```
    UninstallSampleApp theApp = new UninstallSampleApp(true);
    theApp.enterEventDispatcher();
      }
}
```

The constructor receives a flag indicating whether it is running in the foreground, so the initialization tasks can be performed according to this information (that is, no UI is needed for the background process).

Implement the listener. It is called every time a module is about to be removed or added to the system, so the events must be filtered according to the module name.

```
public void moduleDeletionsPending(String[] modules) {
String currentModuleName =
ApplicationDescriptor.currentApplicationDescriptor().getModuleName(
);
SDMConstants constants = SDMConstants.getInstance();
for (int i=0; i < modules.length; i++) {
  if (modules[i].equals(currentModuleName)) {

PersistentStore.destroyPersistentObject(constants.getId(SDMConstant
s.SERVICE_DOC_KEY));

PersistentStore.destroyPersistentObject(constants.getId(SDMConstant
s.METADATA_KEY));

PersistentStore.destroyPersistentObject(constants.getId(SDMConstant
s.DATA_ENTRY_KEY));

PersistentStore.destroyPersistentObject(constants.getId(SDMConstant
s.PREFERENCES_KEY));
    break;
    }
}
```

This example shows how to remove the persisted cache components and the preferences, but any persisted application data can be removed the same way.

The BlackBerry Persistent Store APIs do not provide a relational database model. The application must create an effective object model and manage the relationships between objects as necessary, using indices and hash tables. The keys used to store/load objects must always be handled by the applications. Encryption/decryption is performed with the help of the PersistentContent object. Research In Motion (RIM) must track the use of some sensitive BlackBerry APIs for security and export control reasons. To load your application on a BlackBerry smart phone, the application must be signed using a signature key (provided by RIM). The application owner must order signing keys in order to access the BlackBerry runtime, application and cryptography APIs.

Only the applications that are signed with RIM provided keys can use the persistent store, but there will not be any access control to the persisted data. Any kind of application signed by RIM keys can read and replace your persisted data. If you want to protect your data from other applications, you have to use the BlackBerry Signing Authority Tool to sign the resulting cod file with your private key. If you do not have a private key for signing, you need to use the BlackBerry Signing Authority Admin Tool to create a public/private key pair. In order for your application to access protected persistent content, the developer must set the used signerID in

```
ISDMPreferences.PERSISTENCE_ACCESS_CONTROL_SIGNER_ID
```

preference.

The encryption/decryption in the case of saving a huge number of objects or, for example, a Vector which contains thousands of items can be slow on BlackBerry phones, because the operation must be done on each field of each object. For encryption, the library uses the underlying OS encryption API, no custom API is provided for this purpose. The BlackBerry API offers the `PersistentContent` class for the applications, which can be used to encrypt/decrypt Strings and byte arrays.

SDMConnectivity

The Network layer handles all network layer related tasks, hides the complexity of network communication, and provides easy to use APIs to the applications.

List of Features

- Provides interfaces for request handling
- Handles the requests asynchronously
- Handles the requests by multiple number of threads (configurable)
- Handles the data streaming from server to device and device to server. A streaming API is provided to allow the value to be accessed in chunks. See *SDMRequest* and *SDMResponse*, for more information on streaming APIs.

SDMConnectivity Public APIs

Provides a list of public APIs in the SDMConnectivity library.

Technical Details and SDMConnectivity Public APIs

Note: The SAP Mobile Platform APIs and their descriptions are available after the installation of SAP Mobile Platform at the following location within your installation folder: `...\UnwiredPlatform\ClientAPI\apidoc`.

The `SDMRequestManager` class implements the ISDMRequestManager interface, which provides the following methods:

```
ISDMRequestManager

void makeRequest(final ISDMRequest aRequest);
void makeRequest(final ISDMBundleRequest aBundleRequest)
ISDMConnectivitiyParameters getConnectivityParameters()
Vector getAllRequests()
int getQueueSize()
byte[] getRootContextID()
void terminate()
void pause()
void resume()
```

The number of working threads in the `RequestManager` class is configurable via the `initialize(final SDMConnectivityParameters aParameters, final int aThreadNumber)` method. The number of threads is maximized in four by the connectivity layer, because of performance related issues. If the client initializes the layer with more than the allowed threads, the implementation of the connectivity layer will decrease the thread number to the max allowed number. Methods defined by the `SDMConnectivityParameters` class:

```
ISDMConnectivitiyParameters

void setUserName(String aUserName)
String getUserName()
void setUserPassword(String aPassword)
String getUserPassword()
void setBaseUrl(String baseUrl)
String getBaseUrl()
String getLanguage()
void setLanguage(String language)
void enableXSRF(boolean useXSRF)
```

Sending requests with the connectivity layer consists of the following steps:

1. Create the `RequestManager` class and initialize it with the required parameters.
2. Create the request object. This can be done by implementing the ISDMRequest interface or by extending the `SDMBaseRequest` class which is the base implementation of the ISDMRequest interface. Both of them are provided by the connectivity layer.
3. Add the request object to the `SDMRequestManager`.

```
Example

//create and fill parameters for Connectivity library
SDMConnectivityParameters params = new SDMConnectivityParameters();
params.setUserName("test");
params.setUserPassword("testpwd");
params.setLogger(Logger.getInstance()); //get the default Logger
//create the RequestManager
SDMRequestManager reqManager = new SDMRequestManager();
//initialize it
reqManager.initialize(params, 2);//set the parameters and the thread
number to be used
//create the request object
ISDMRequest testRequest = new SDMBaseRequest();
testRequest.setRequestUrl("http://test.de:8080/testpath");
testRequest.setRequestMethod(ISDMRequest.REQUEST_METHOD_GET);
testRequest.setPriority(ISDMRequest.PRIORITY_NORMAL);
//add the request to the connectivity layer
reManager.makeRequest(testRequest);
```

The tasks of the connectivity library have been divided into three main categories: managing the request queues, managing reading and writing to the input/output streams, and managing the platform specific connection creation.

The Connectivity component always performs the requests in asynchronous mode. The application's role is to handle the request in sync mode. The component is able to perform HTTP and HTTPS requests, which you can use for developing and testing purposes, but the default is SUPRequest. The threads in the connectivity library are responsible for taking the requests from the queue (FIFO - First in first out - algorithm) and performing the requests.

The number of working threads in the connection pool can be configured in the connectivity layer. There is only one queue, and this is handled by the `SDMRequestManager`, and the working threads take the requests from this queue. Applications are interacting only with the `SDMRequestManager` class; the other components of the connectivity library are not visible to them. The network component consists of three main parts:

- `SDMRequestManager`: responsible for queuing the requests, managing the threads and keeping the connection with applications
- `ConnectionHandler`: responsible for performing the request
- `ConnectionFactory`: responsible for creating and managing platform dependent connections to the server

An application can have more than one `SDMRequestManager`, for example, when connecting to two different servers at the same time.

There is built-in support for setting the timeout for the socket connection, the application can use the `SDMConnectivityParameters` object to modify the value.

```
int TIMEOUT = 3500;

ISDMPreferences preferences = new SDMPreferences();

preferences.setPreference(ISDMPreferences.CONNECTION_TIMEOUT_MS,
String.valueOf(TIMEOUT));

requestManager = new SDMRequestManager(logger, preferences,
parameters, NUM_OF_HTTP_EXECUTION_THREADS);
```

SDMRequest

An SDMRequest object wraps all the information which is needed by the connectivity library to be able to perform the requests. The connectivity library interacts with the request object to query the necessary information about the headers, the post data, and so on.

The connectivity layer also uses the request object to notify the application about the result of the request using the ISDMNetListener interface. The connectivity component provides an interface called the ISDMRequest and a base implementation of it called the SDMBaseRequest. The applications have to extend this base class when creating new application specific requests. The ISDMRequest interface defines the following public APIs:

```
ISDMRequest

void setRequestUrl(final String aUrl)
String getRequestUrl()
void setRequestMethod(final int aRequestMethod)
int getRequestMethod()
```

```
byte[] getData()
void setPriority(final int aPriority)
int getPriority()
boolean useCookies()
void setListener(final ISDMNetListener aListener)
ISDMNetListener getListener()
boolean hasPostData()
void postData(OutputStream os)
void setHeaders(final Hashtable aHashtable)
Hashtable getHeaders()
void appendHeaders(final Hashtable aHashtable)
void appendHeader(final String aHeaderName, final String
aHeaderValue)
void setEtag(String eTag, int eTagHeader)
void enableStreaming(boolean tobestreamed)
boolean isStreamingEnabled()
InputStream getDataStream()
void setDataStream(InputStream iStream)
```

The `enableStreaming` API is used by the application to enable streaming for downloading large amount of data from server to device.

The `SetDataStream` API is used by the application to enable streaming for uploading large amount of data from device to server.

The ISDMNetListener interface can be used by the client to be notified by the connectivity layer about the result of a request. Usage of this feature is not mandatory, however, you can handle incidental errors with it. Methods available in the ISDMNetListener interface:

```
ISDMNetListener

void onSuccess(ISDMRequest aRequest, IHttpResponse aResponse)
void onError(ISDMRequest aRequest, ISDMResponse aResponse,
ISDMRequestStateElement aRequestStateElement)
```

The role of the `SDMRequestStateElement` object used by the connectivity library is to provide the application with more detail on the occurred error. Methods available in `SDMRequestStateElement` object:

```
ISDMRequestStateElement

int getErrorCode()
void setErrorCode(final int code)
int getHttpStatusCode()
void setHttpStatusCode(final int httpStatus)
Exception getException()
void setException(final Exception aException)
String getRedirectLocation()
IHttpResponse getResponse()
```

```
Example for successful response received by application from EIS

     public void onSuccess(ISDMRequest  aRequest, SDMHttpResponse
aResponse) {
     System.out.println("Http response status code:" +
aResponse.getStatusCode());
     System.out.println("Cookie string:" + aResponse.getCookie());
     byte[] content = aResponse.getContent();
     String response = new String(content);
     System.out.println("Received content:" + response);
     //get the headers
     Hashtable headers = aResponse.getHeaders();
}
```

```
Example for data streaming request made by application

     final ISDMRequest request = new SDMBaseRequest();

     Hashtable headers = new Hashtable();
     headers.put("Content-Type", "application/atom+xml");
     request.setHeaders(headers);
     request.setPriority(ISDMRequest.PRIORITY_HIGH);
     request.setRequestMethod(ISDMRequest.REQUEST_METHOD_GET);
     request.setRequestUrl(url);
     request.setListener(listener);
     request.enableStreaming(true);
     requestManager.makeRequest(request);
}
```

SDMResponse

An `SDMResponse` object wraps all the information needed by the connectivity library to be respond back to the application.

The connectivity component provides an interface called ISDMResponse. Available SDMResponse APIs are:

```
ISDMResponse

    boolean isDataAvailable();
    String getCookie();
    int getCurrentOffset();
    void setCorelationId(String coRelationId);
    String getCorelationId();
    int getDataStream(byte[] buffer, int chunkSize);
    int getDataStream(byte[] buffer, int chunkSize, int offset, 
String cookie);
```

The `getDataStream` API is used by the application to retrieve the data stream. This API returns the length of data stream, and the application invokes this API until the length of buffer stream is equal to 0.

```
Example for successful streaming response received by application

public void onSuccess(ISDMRequest aRequest, IHttpResponse 
aResponse)
{
      SUPResponse resp = (SUPResponse)aResponse;
      int buffersize = 102400;
      byte[] buf = new byte[buffersize];
      int bytesRecvd,j=0,i=0;
```

```
        try {
                    while((bytesRecvd = 
resp.getDataStream(buf,buffersize))>0)

{
                                            //Same as Andriod
                           String respdata = new String(buf).trim();

System.out.println("BytesReceived"+bytesRecvd);
...
...
...
        buf = new byte[buffersize];
       }
} catch (Exception e) {
  // TODO Auto-generated catch block
       e.printStackTrace();
  }
 }
}
```

ETag Support

The SDMConnectivity library provides APIs that allow ETags to be forwarded as request headers.

HTTP header fields are components of the message header for requests and responses. These fields define the the operating parameters of an HTTP transaction. The following header fields are applicable for ETag support.

Table 8. ETag Header Fields

Header Fields	Description
If-Match	A client that has one or more ETags obtained from a resource, can verify that one of these ETags is current.
If-None-Match	A client that has one or more ETags obtained from a resource can verify that none of these ETags is current.
If-Range	A client that has an ETag that matches the current ETag of a resource, can request for the specified sub-range of the resource. This means that the client has a partial copy of an entity in its cache and can request to get the entire entity.

Note: If the backend system corresponds to Gateway, only the If-Match header field is supported.

Using ETags for HTTP Request Methods

ETags are forwarded as headers in HTTP request methods. Here are some examples for ETags with some of the commonly used request methods.

- HTTP GET

  This method requests a representation of the specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the value of the specified resource on the server is returned. If it fails, the status code returned is 412 (Precondition Failed)
- HTTP PUT

  This method uploads the representation of the specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the specified resource is uploaded at the server. If it fails, the status code returned is 412 (Precondition Failed).
- HTTP DELETE

  This method deletes a specified resource. If used with an If-Match header and if the ETag of the entry resource matches the ETag at the server, the resource is deleted. If it fails, the status code returned is 412 (Precondition Failed).

SDMConfiguration

Each low level API has its own defaults/constants set in the SDMConfiguration library. Default values of preferences can be found in the SDMConstants class.

List of Features

- Providing modifiable preferences for SDMComponent libraries
- Encrypting/decrypting values of preferences for persistence
- Providing API for resetting the preferences of SDMComponent libraries to their default values
- Providing API for creating and handling custom preferences
- Notifying subscribed listeners in case of any change in preferences

SDMConfiguration Public APIs

Provides a list of public APIs in the SDMConfiguration library.

SDMConfiguration Public APIs

```
ISDMPreferences

void setPreference(String key, String value)
String getPreference(String key)
void registerPreferenceChangeListener(String key,
ISDMPreferenceChangeListener changeListener)
void unRegisterPreferenceChangeListener(String key,
ISDMPreferenceChangeListener changeListener)
Hashtable encrypt()
Hashtable decrypt()
void initFromPersistence(Hashtable prefs)
void deletePreference(final String aKey)
void reset()
```

Technical Details

SDMPreferences object is used for storing configuration key-value pairs. Only the String representation of the value can be stored. Persistent storage of this object is available from SDMPersistence. This object calls `encrypt()`, `decrypt()` and `initFromPersistence()` methods of SDMPreferences, so the applications do not have to use these methods explicitly.

During instantiation of SDMPreference, the default values needed for other SDMComponents are filled. SDMComponents preferences can be reset to their default values using the `reset()` method.

You can register a preference change listener for each preference in SDMPreferences (including custom preferences) so that you will be notified if the value of a given preference has changed. Preference change listener notification and preference validation can only be done after the initialization of the appropriate component.

SDMSupportability

The OData SDK provides a set of features and concepts for the supportability of the applications built on top of the SDK.

SDMLogger

The SDMLogger architecture follows the logging implementation in Java 1.5 and provides the same services and structures, but also contains BlackBerry and OData SDK specific implementations.

The component provides the following features:

- Filtering: the client app can set the log level. Provides filterable log retrieval by component and by timestamp (from-to).
- Formatting: before the log message is sent to the handler (which performs the logging), there is a possibility to format the message.
- Handlers: handlers are responsible for logging the messages to the specified place. Depending on the implementation of the handler, the place can be the memory, a file, or the message can be sent to the server. Changing the default handlers in the Logger implementation is invisible for the client.

Current implementation contains implementation for all the interfaces (the IFilter, IHandler and IFormatter). These classes begin with the “Default” prefix.

SDMLogger Public APIs

```
ISDMLogger
```

```
ISDMPreferences getPreferences()
void entering(String sourceClass, String sourceMethod)
void entering(String sourceClass, String sourceMethod, Object
param1)
void entering(String sourceClass, String sourceMethod,
            Object[] params)
void exiting(String sourceClass, String sourceMethod)
void exiting(String sourceClass, String sourceMethod, Object result)
void fine(String msg)
void finer(String msg)
void info(String msg)
void log(final int level, String msg)
void log(final int level, String msg, final Object param1)
void log(final int level, String msg, Object[] params)
void log(final int level, String msg, Throwable thrown)
void log(final int level, final String message, final Exception ex)
void logNestedObjects(final int level, String message,
            final Object[] params)
void setHandler(IHandler handler)
void error(String msg)
void p(final String message, long timestamp)
Vector getLogRecords()
Vector getLogRecorsdByComponentName(final String componentName)
Vector getLogRecorsdByTimeStamp(final long start, final long end)
void clearLogRecords()
int getLogNumber()
String getLogHeader()
```

SAP Passport

For the Single Activity Trace an SAP Passport has to be issued by the connectivity layer of the library.

The SAP Passport is transported as an HTTP header in the request. The server handles the SAP Passport to generate end-to-end Trace. The OData SDK is using JSDR SAP Passport sources integrated in the library at source level. It can be turned on or off with `ISDMPreferences.SDM_TRACEING_ENABLED` preference key. By default it is turned off.

ODP SDK API Usage

The ODP SDK consists of APIs used to customize applications to send and receive data using Online Data Proxy.

For a comprehensive list of API references, extract the contents from the following zip files:

- *SMP_HOME*`\MobileSDK<Version>\OData\BB\docs\SUPProxyClient-<version>-docs.zip`
- *SMP_HOME*`\MobileSDK<Version>\OData\BB\docs\BBODataSDK-<version>-docs.zip`

See also

- *OData SDK Components and APIs* on page 329

Security APIs

The security APIs allow you to customize some aspects of secure storage.

DataVault

The DataVault class provides encrypted storage of occasionally used, small pieces of data. All exceptions thrown by DataVault methods are of type DataVaultException.

By adding the MCL.jar library to your project, you can use the DataVault class for on-device persistent storage of certificates, database encryption keys, passwords, and other sensitive items. Use this class to:

- Create a vault
- Set a vault's properties
- Store objects in a vault
- Retrieve objects from a vault
- Change the password used to access a vault

The contents of the data vault are strongly encrypted using AES-256. The DataVault class allows you create a named vault, and specify a password and salt used to unlock it. The password can be of arbitrary length and can include any characters. The password and salt together generate the AES key. If the user enters the same password when unlocking, the contents are decrypted. If the user enters an incorrect password, exceptions occur. If the user enters an incorrect password a configurable number of times, the vault is deleted and any data stored within it becomes unrecoverable. The vault can also relock itself after a configurable amount of time.

Typical usage of the `DataVault` is to implement an application login screen. Upon application start, the user is prompted for a password, which unlocks the vault. If the unlock attempt is successful, the user is allowed into the rest of the application. User credentials for synchronization can also be extracted from the vault so the user need not reenter passwords.

createVault

Creates a new secure store (a vault)

A unique name is assigned, and after creation, the vault is referenced and accessed by that name. This method also assigns a password and salt value to the vault. If a vault with the same name already exists, this method throws an exception. A newly created vault is in the unlocked state.

Syntax

```
public static DataVault createVault(
   String name,
   String password,
   String salt
)
```

Parameters

- **name –** an arbitrary name for a `DataVault` instance on this device. This name is effectively the primary key for looking up `DataVault` instances on the device, so it cannot use the same name as any existing instance. If it does, this method throws an exception with error code INVALID_ARG. The name also cannot be empty or null.
- **password –** the initial encryption password for this `DataVault`. This is the password needed for unlocking the vault. If null is passed, a default password is computed and used.
- **salt –** the encryption salt value for this `DataVault`. This value, combined with the password, creates the actual encryption key that protects the data in the vault. If null is passed, a default salt is computed and used.

Returns

Returns the newly created instance of the DataVault with the provided ID. The returned DataVault is in the unlocked state with default configuration values. To change the default configuration values, you can immediately call the "set" methods for the values you want to change.

Examples

- **Create a data vault –** creates a new data vault called `myVault`.

  ```
  DataVault vault = null;
  if (!DataVault.vaultExists("myVault"))
  {
     vault = DataVault.createVault("myVault", "password", "salt");
  }
  else
  {
     vault = DataVault.getVault("myVault");
  }
  ```

vaultExists

Tests whether the specified vault exists.

Syntax

```
public static boolean vaultExists(String name)
```

Parameters

- **name –** the vault name.

Returns

Returns true if the vault exists; otherwise returns false.

Examples

- **Check if a data vault exists –** checks if a data vault called `myVault` exists, and if so, deletes it.

  ```
  if (DataVault.vaultExists("myVault"))
  {
     DataVault.deleteVault("myVault");
  }
  ```

getVault

Retrieves a vault.

Syntax

```
public static DataVault getVault(String name)
```

Parameters

- **name –** the vault name.

Returns

Returns a `DataVault` instance.

If the vault does not exist, a `DataVaultException` is thrown.

deleteVault

Deletes the specified vault from on-device storage.

Note: When you have a shared data vault, if one application deletes the vault, the data vault is no longer accessible in the other application.

If the vault does not exist, this method throws an exception. The vault need not be in the unlocked state, and can be deleted even if the password is unknown.

Syntax

```
public static void deleteVault(String name)
```

Parameters

- **name –** the vault name.

Examples

- **Delete a data vault –** deletes a data vault called `myVault`.

  ```
  if (DataVault.vaultExists("myVault"))
  {
     DataVault.deleteVault("myVault");
  }
  ```

getDataNames

Retrieves information about the data names stored in the vault.

The application can pass the data names to `getValue` or `getString` to retrieve the data values.

Syntax

```
public abstract DataVault.DVDataName[] getDataNames()
```

Parameters

None.

Returns

Returns a `DVPasswordPolicy` object, as an array of `DVDataName` structure objects.

Examples

- **Get data names**

  ```
  // Call getDataNames to retrieve all stored element names from our
  data vault.
  DataVault.DVDataName[] dataNameArray = oDataVault.getDataNames();
  for ( int i = 0; i < dataNameArray.length; i++ )
  {
    if ( dataNameArray[i].iType == DataVault.DV_DATA_TYPE_STRING )
    {
      String thisStringValue =
  oDataVault.getString( dataNameArray[i].sName );
    }
    else
    {
      byte[] thisBinaryValue =
  oDataVault.getValue( dataNameArray[i].sName );
    }
  }
  ```

setPasswordPolicy

Stores the password policy and applies it when `changePassword` is called, or when validating the password in the `unlock` method.

If the application has not set a password policy using this method, the data vault does not validate the password in the `createVault` or `changePassword` method. An exception is thrown if there is any invalid (negative) value in the `passwordPolicy` object.

Syntax

```
public abstract void setPasswordPolicy(DataVault.DVPasswordPolicy
oPasswordPolicy)
```

Parameters

- **oPasswordPolicy –** the password policy constraints.

Returns

None.

Examples

- **Set a password policy**

  ```
  // SetPasswordPolicy() locks the vault to ensure the old password
  // conforms to the new password policy settings.
  oDataVault.setPasswordPolicy( oPasswordPolicy );
  ```

Password Policy Structure

A structure defines the policy used to generate the password.

Table 9. Password Policy Structure

Name	Type	Description
defaultPasswordAllowed	Boolean	Indicates if client application is allowed to use default password for the data Vault. If this is set to TRUE and if client application uses default password then minLength, hasDigits, hasUpper, hasLower and hasSpecial parameters in the policy are ignored.

Name	Type	Description
minimumLength	Integer	The minimum length of the password.
hasDigits	Boolean	Indicates if the password must contain digits.
hasUpper	Boolean	Indicates if the password must contain uppercase characters.
hasLower	Boolean	Indicates if the password must contain lowercase characters.
hasSpecial	Boolean	Indicates if the password must contain special characters. The set of special characters is: “~!@#$%^&*()-+”.
expirationDays	Integer	Specifies password expiry days from the date of setting the password. 0 indicates no expiry.
minUniqueChars	Integer	The minimum number of unique characters in the password. For example, if length is 5 and minUniqueChars is 4 then “aaate” or “ababa” would be invalid passwords. Instead, “aaord” would be a valid password.
lockTimeout	Integer	The timeout value (in seconds) after which the vault will be locked from the unlock time. 0 indicates no timeout. This value overrides the value set by `setLockTimeout` method.
retryLimit	Integer	The number of failed unlock attempts after which data vault is deleted. 0 indicates no retry limit. This value overrides the value set by the `setRetryLimit` method.

Settings for Password Policy

The client applications use these settings to fill the PasswordPolicy structure. The default values are used by the data vault when no policy is configured. The defaults are also used in SAP Control Center in the default template. The SAP Mobile Platform administrator can modify these settings through SAP Control Center. The application must set the password policy for the data vault with the administrative (or alternative) settings.

Note: Setting the password policy locks the vault. The password policy is enforced when `unlock` is called (because the password is not saved, calling `unlock` is the only time that the policy can be evaluated).

- **MCL_PROP_ID_PWDPOLICY_ENABLED –** Boolean property with a default value of false. Indicates if a password policy is enabled by the administrator.
- **MCL_PROP_ID_PWDPOLICY_DEFAULT_PASSWORD_ALLOWED –** Boolean property with a default value of false. Indicates if the client application is allowed to use the default password for the data vault.
- **MCL_PROP_ID_PWDPOLICY_MIN_LENGTH –** Integer property with a default value of 0. Defines the minimum length for the password.
- **MCL_PROP_ID_PWDPOLICY_HAS_DIGITS –** Boolean property with a default value of false. Indicates if the password must contain digits.
- **MCL_PROP_ID_PWDPOLICY_HAS_UPPER –** Boolean property with a default value of false. Indicates if the password must contain at least one uppercase character.
- **MCL_PROP_ID_PWDPOLICY_HAS_LOWER –** Boolean property with a default value of false. Indicates if the password must contain at least one lowercase character.
- **MCL_PROP_ID_PWDPOLICY_HAS_SPECIAL –** Boolean property with a default value of false. Indicates if the password must contain at least one special character. A special character is a character in this set “~!@#$%^&*()-+”.
- **MCL_PROP_ID_PWDPOLICY_EXPIRATION_DAYS –** Integer property with a default value of 0. Specifies the number of days in which password will expire from the date of setting the password. Password expiration is checked only when the vault is unlocked.
- **MCL_PROP_ID_PWDPOLICY_MIN_UNIQUE_CHARS –** Integer property with a default value of 0. Specifies minimum number of unique characters in the password. For example, if minimum length is 5 and minUniqueChars is 4 then “aaate” or “ababa” would be invalid passwords. Instead, “aaord” would be a valid password.
- **MCL_PROP_ID_PWDPOLICY_LOCK_TIMEOUT –** Integer property with a default value of 0. Specifies timeout value (in seconds) after which the vault is locked from the unlock time. 0 indicates no timeout.
- **MCL_PROP_ID_PWDPOLICY_RETRY_LIMIT –** Integer property with a default value of 0. Specifies the number of failed unlock attempts after which data vault is deleted. 0 indicates no retry limit.

Password Errors

Password policy violations cause exceptions to be thrown.

Table 10. Password Errors

Name	Value	Description
PASSWORD_REQUIRED	50	Indicates that a blank or null password was used when the password policy does not allow default password.
PASSWORD_UNDER_MIN_LENGTH	51	Indicates that the password length is less than the required minimum.
PASSWORD_REQUIRES_DIGIT	52	Indicates that the password does not contain digits.
PASSWORD_REQUIRES_UPPER	53	Indicates that the password does not contain upper case characters.
PASSWORD_REQUIRES_LOWER	54	Indicates that the password does not contain lower case characters.
PASSWORD_REQUIRES_SPECIAL	55	Indicates that the password does not contain one of these special characters: ~!@#$%^&*()-+.
PASSWORD_UNDER_MIN_UNIQUE	56	Indicates that the password contains fewer than the minimum required number of unique characters.
PASSWORD_EXPIRED	57	Indicates that the password has been in use longer than the number of configured expiration days.

getPasswordPolicy android blackberry

Retrieves the password policy set by `setPasswordPolicy`.

Use this method once the `DataVault` is unlocked.

Syntax

```
public abstract DataVault.DVPasswordPolicy getPasswordPolicy()
```

Parameters

None.

Returns

Returns a `passwordPolicy` structure that contains the policy set by `setPasswordPolicy`.

Returns a `DVPasswordPolicy` object with the default values if no password policy is set.

Examples

- **Get the current password policy**

```
// Call getPasswordPolicy() to return the current password policy
settings.
      DataVault.DVPasswordPolicy oCurrentPolicy =
oDataVault.getPasswordPolicy();
```

isDefaultPasswordUsed

Checks whether the default password is used by the vault.

Use this method once the `DataVault` is unlocked.

Syntax

```
public boolean isDefaultPasswordUsed()
```

Parameters

None.

Returns

Returns	Indicates
true	Both the default password and the default salt are used to encrypt the vault.
false	Either the default password or the default salt is not used to encrypt the vault.

Examples

- **Check if default password used**

```
 // Call isDefaultPasswordused() to see if we are using an
automatically
```

```
// generated password (which we are).
boolean isDefaultPasswordUsed =
oDataVault.isDefaultPasswordUsed();
```

This code example lacks exception handling. For a code example that includes exception handling, see *Developer Guide: BlackBerry Object API Applications> Client Object API Usage > Security APIs > DataVault > Code Sample*.

lock

Locks the vault.

Once a vault is locked, you must unlock it before changing the vault's properties or storing anything in it. If the vault is already locked, `lock` has no effect.

Syntax

```
public void lock()
```

Parameters

None.

Returns

Examples

- **Locks the data vault –** prevents changing the vaults properties or stored content.

  ```
  vault.lock();
  ```

isLocked

Checks whether the vault is locked.

Syntax

```
public boolean isLocked()
```

Parameters

None.

Returns

Returns	Indicates
true	The vault is locked.
false	The vault is unlocked.

unlock

Unlocks the vault.

Unlock the vault before changing the its properties or storing anything in it. If the incorrect password or salt is used, this method throws an exception. If the number of unsuccessful attempts exceeds the retry limit, the vault is deleted.

The password is validated against the password policy if it has been set using `setPasswordPolicy`. If the password is not compatible with the password policy, an `IncompatiblePassword` exception is thrown. In that case, call `changePassword` to set a new password that is compatible with the password policy.

Syntax

```
public void unlock(String password, String salt)
```

Parameters

- **password –** the initial encryption password for this `DataVault`. If null is passed, a default password is computed and used.
- **salt –** the encryption salt value for this `DataVault`. This value, combined with the password, creates the actual encryption key that protects the data in the vault. If null is passed, a default salt is computed and used.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `INVALID_PASSWORD`.

Examples

- **Unlocks the data vault –** once the vault is unlocked, you can change its properties and stored content.

  ```
  if (vault.isLocked())
  {
     vault.unlock("password", "salt");
  }
  ```

setString

Stores a string object in the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public void setString(
   String name,
```

```
   String value
)
```

Parameters

- **name –** the name associated with the string object to be stored.
- **value –** the string object to store in the vault.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `INVALID_PASSWORD`.

Examples

- **Set a string value –** creates a test string, unlocks the vault, and sets a string value associated with the name `"testString"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

  ```
  string teststring = "ABCDEFabcdef";
  try
  {
     vault.unlock("password", "salt");
     vault.setString("testString", teststring);
  }
  catch (DataVaultException e)
  {
     System.out.println("Exception: " + e.toString());
  }
  finally
  {
     vault.lock();
  }
  ```

getString

Retrieves a string value from the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public String getString(String name)
```

Parameters

- **name –** the name associated with the string object to be retrieved.

Returns

If an incorrect password or salt is used, a `DataVaultException` is thrown with the reason `INVALID_PASSWORD`.

Examples

- **Get a string value –** unlocks the vault and retrieves a string value associated with the name `"testString"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
try
{
   vault.unlock("password", "salt");
   string retrievedstring = vault.getString("testString");
}
catch (DataVaultException e)
{
   System.out.println("Exception: " + e.toString());
}
finally
{
   vault.lock();
}
```

setValue

Stores a binary object in the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public void setValue(
   string name,
   byte[] value
)
```

Parameters

- **name –** the name associated with the binary object to be stored.
- **value –** the binary object to store in the vault.

Returns

None.

Examples

- **Set a binary value –** unlocks the vault and stores a binary value associated with the name `"testValue"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
try
{
   vault.unlock("password", "salt");
   vault.setValue("testValue", new byte[] { 1, 2, 3, 4, 5});
}
catch (DataVaultException e)
{
   System.out.println("Exception: " + e.toString());
}
finally
{
   vault.lock();
}
```

getValue

Retrieves a binary object from the vault.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public byte[] getValue(string name)
```

Parameters

- **name –** the name associated with the binary object to be retrieved.

Returns

None.

Examples

- **Get a binary value –** unlocks the vault and retrieves a binary value associated with the name `"testValue"` in the vault. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
try
{
   vault.unlock("password", "salt");
   byte[] retrievedvalue = vault.getValue("testValue");
}
catch (DataVaultException e)
{
   System.out.println("Exception: " + e.toString());
}
finally
{
   vault.lock();
}
```

deleteValue

Deletes the specified value.

An exception is thrown if the vault is locked when this method is called.

Syntax

```
public static void deleteValue(String name)
```

Parameters

- **name –** the name of the value to be deleted.

Returns

None.

Examples

- **Delete a value –** deletes a value called `myValue`.

  ```
  DataVault.deleteValue("myValue");
  ```

changePassword (two parameters)

Changes the password for the vault. Use this method when the vault is unlocked.

Modifies all name/value pairs in the vault to be encrypted with a new password/salt. If the vault is locked or the new password is empty, an exception is thrown.

Syntax

```
public void changePassword(
   String newPassword,
   String newSalt
)
```

Parameters

- **newPassword –** the new password.
- **newSalt –** the new encryption salt value.

Returns

None.

Examples

- **Change the password for a data vault –** changes the password to `"newPassword"`. The `finally` clause in the `try/catch` block ensures that the vault ends in a secure state even if an exception occurs.

```
try
{
   vault.unlock("password", "salt");
   vault.changePassword("newPassword", "newSalt");
}
catch (DataVaultException e)
{
   System.out.println("Exception: " + e.toString());
}
finally
{
   vault.lock();
}
```

changePassword (four parameters)

Changes the password for the vault. Use this method when the vault is locked

This overloaded method ensures the new password is compatible with the password policy, uses the current password to unlock the vault, and changes the password of the vault to a new password. If the current password is not valid an `InvalidPassword` exception is thrown. If the new password is not compatible with the password policy set in `setPasswordPolicy` then an `IncompatiblePassword` exception is thrown.

Syntax

```
public abstract void changePassword(string sCurrentPassword,
    string sCurrentSalt,
    string sNewPassword,
    string sNewSalt)
```

Parameters

- **currentPassword –** the current encryption password for this data vault. If a null value is passed, a default password is computed and used.
- **currentSalt –** the current encryption salt value for this data vault. If a null value is passed, a default password is computed and used.
- **newPassword –** the new encryption password for this data vault. If a null value is passed, a default password is computed and used.
- **newSalt –** the new encryption salt value for this data vault. This value, combined with the password, creates the actual encryption key that protects the data in the vault. This value may be an application-specific constant. If a null value is passed, a default password is computed and used.

Returns

None.

Examples

- **Change the password for a data vault**

```
// Call changePassword with four parameters, even if the vault is
locked.
// Pass null for oldSalt and oldPassword if the defaults were
used.
oDataVault.changePassword( null, null, "password!1A",
"saltD#ddg#k05%gnd[!1A" );
```

Code Sample

Create a data vault for encrypted storage of application data.

```
public void testFunctionality()
{
   try
   {
      DataVault oDataVault = null;

      // If this dataVault already exists, then get it by calling
getVault()
      // Else create this new dataVault by calling createVault()
      if ( DataVault.vaultExists( "DataVaultExample" ) )
         oDataVault = DataVault.getVault( "DataVaultExample" );
      else
         oDataVault = DataVault.createVault( "DataVaultExample",
"password!1A", "saltD#ddg#k05%gnd[!1A" );

     // Call setLockTimeout(). This allows you to set the timeout of
the vault in seconds
      oDataVault.setLockTimeout( 1500 );
      int iTimeout = oDataVault.getLockTimeout();

      // Call setRetryLimit(). This allows you to set the number of
retries before the vault is destroyed
      oDataVault.setRetryLimit( 10 );
      int iRetryLimit = oDataVault.getRetryLimit();

      // Call setPasswordPolicy(). The passwordPolicy also includes
the retryLimit and LockTimeout that we set above.
      DataVault.DVPasswordPolicy oPasswordPolicy = new
DataVault.DVPasswordPolicy();
      oPasswordPolicy.bDefaultPasswordAllowed   = true;
      oPasswordPolicy.iMinLength                = 4;
      oPasswordPolicy.bHasDigits                = true;
      oPasswordPolicy.bHasUpper                 = true;
      oPasswordPolicy.bHasLower                 = true;
      oPasswordPolicy.bHasSpecial               = true;
      oPasswordPolicy.iExpirationDays           = 20;
      oPasswordPolicy.iMinUniqueChars           = 3;
      oPasswordPolicy.iLockTimeout              = 1600;
      oPasswordPolicy.iRetryLimit               = 20;

     // SetPasswordPolicy() will always lock the vault to ensure the
old password
```

```
      // conforms to the new password policy settings.
      oDataVault.setPasswordPolicy( oPasswordPolicy );

     // We are now locked and need to unlock before we can access the
vault.
      oDataVault.unlock( "password!1A", "saltD#ddg#k05%gnd[!1A" );

      // Call getPasswordPolicy() to return the current password
policy settings.
      DataVault.DVPasswordPolicy oCurrentPolicy =
oDataVault.getPasswordPolicy();

      // Call setString() by giving it a name:value pair to encrypt
and persist
      // a string data type within your dataVault.
      oDataVault.setString( "stringName", "stringValue" );

      // Call getString to retrieve the string we just stored in our
data vault!
      String storedStringValue =
oDataVault.getString( "stringName" );

     // Call setValue() by giving it a name:value pair to encrypt and
persist
      // a binary data type within your dataVault.
      byte[] binaryValue = { 1, 2, 3, 4, 5, 6, 7  };
      oDataVault.setValue( "binaryName", binaryValue );

      // Call getValue to retrieve the binary we just stored in our
data vault!
     byte[] storedBinaryValue = oDataVault.getValue( "binaryName" );

      // Call getDataNames to retrieve all stored element names from
our data vault.
      DataVault.DVDataName[] dataNameArray =
oDataVault.getDataNames();
      for ( int i = 0; i < dataNameArray.length; i++ )
      {
         if ( dataNameArray[i].iType ==
DataVault.DV_DATA_TYPE_STRING )
         {
            String thisStringValue =
oDataVault.getString( dataNameArray[i].sName );
         }
         else
         {
            byte[] thisBinaryValue =
oDataVault.getValue( dataNameArray[i].sName );
         }
      }

      // Call changePassword with 2 parameters. Vault must be
unlocked.
     // If you pass null parameters as your new password or your new
salt,
      // it will generate a default password or default salt,
respectively.
      oDataVault.changePassword( null, null );

      // Call isDefaultPasswordused() to see if we are using an
automatically
      // generated password (which we are).
      boolean isDefaultPasswordUsed =
oDataVault.isDefaultPasswordUsed();

      // Lock the vault.
      oDataVault.lock();

      // Call changePassword with 4 parameters even if the vault is
locked.
      // Here, we pass null for oldSalt and oldPassword because
defaults were used.
      oDataVault.changePassword( null, null, "password!1A",
"saltD#ddg#k05%gnd[!1A" );

     // Call isDefaultPasswordused() and we will see that the default
password is NOT used anymore.
      isDefaultPasswordUsed = oDataVault.isDefaultPasswordUsed();
   }
   catch( Exception exception )
   {

   }
   finally
   {
     // Because this is a test example, we will delete our vault at
the end.
     // This means we will forever lose all data we persisted in our
data vault.
      if ( DataVault.vaultExists( "DataVaultExample" ) )
         DataVault.deleteVault( "DataVaultExample" );
   }
}
```

ODP SDK API Reference for BlackBerry

Use the ODP SDK API reference as the primary reference for all API listings and error code information.

Refer to the ODP SDK API reference for each available package.

client package

Members

All public members of the client package.

- **AppSettingsStore class –** Consists of the methods used for application specific persistent store.

- **ODPAppSettings class –** Consists of methods used to retrieve the setting details required by the application.
- **ODPCertificateManager class –** Consists of methods used to provide the certificate store.
- **ODPClientConnection class –** Consists of the methods used for client-server connection.
- **ODPClientListeners class –** Consists of the methods used by application to get notification from various events and respond to them.
- **ODPException class –** This Class represents the exception thrown by the client library.
- **ODPUserManager class –** Single entry point for all user on-boarding interactions.

AppSettingsStore class

Consists of the methods used for application specific persistent store.

Syntax

```
public class  AppSettingsStore
```

deleteStore() method

Delete the persistent object which is used as application specific store.

Syntax

```
void deleteStore ()
```

Examples

- **Example 1**

```
AppSettingsStore.deleteStore();
```

getProperty(String) method

Retrieve the value associated with a certain application properties.

Syntax

Object getProperty (String *key*)

Parameters

- **key –** Key of the object to be retrieved.

Returns

Application property

Examples

- **Example 1**

  ```
  Object o=getProperty(key);
  ```

setProperty(String, Object) method

Store the application properties as key value pairs.

Syntax

`void setProperty ( String` *key* `, Object` *value* `)` throws `ODPException`

Parameters

- **key –** Key of the object to be retrieved.
- **value –** Value to be stored in association with the key.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

  ```
  AppSettingsStore.setProperty(key,value);
  ```

ODPAppSettings class

Consists of methods used to retrieve the setting details required by the application.

Syntax

`public class ODPAppSettings`

getApplicationEndPoint() method

Retrieve the application end-point with which you can access the business data.

Syntax

`String getApplicationEndPoint ()` throws `ODPException`

Returns

Application end-point

Exceptions

- **ODPException. –**

Examples

- **Example 1**

```
String endpoint=ODPAppSettings.getApplicationEndPoint();
```

getFarmID() method

Retrieve the farm ID provisioned in the client repository.

Syntax

`String` `getFarmID ()` throws `ODPException`

Returns

Farm ID

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
String fg=ODPAppSettings.getFarmID();
```

getPasswordPolicy() method

Retrieve the data vault password policy.

Syntax

`DVPasswordPolicy` `getPasswordPolicy ()` throws `ODPException`

Returns

Password policy

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
DVPasswordPolicy dvp = ODPAppSettings.getPasswordPolicy();
```

getPortNumber() method

Retrieve the port number provisioned in the client repository.

Syntax

`int getPortNumber ()` throws `ODPException`

Returns

Port number

Exceptions

- **ODPException class –**

Examples

- **Example 1**

  ```
  int g=ODPAppSettings.getPortNumber();
  ```

getPushEndPoint() method

Retrieve the push end-point which the EIS can use to push data to the ODP client.

Syntax

`String getPushEndPoint ()` throws `ODPException`

Returns

Push end-point

Exceptions

- **ODPException class –**

Examples

- **Example 1**

  ```
  String pushendpoint=ODPAppSettings.getPushEndPoint();
  ```

getServer() method

Retrieve the SAP Mobile Server name provisioned in the client repository.

Syntax

`String getServer ()` throws `ODPException`

Returns
SAP Mobile Server name

Exceptions

- **ODPException class –**

Examples

- **Example 1**

  ```
  String server=ODPAppSettings.getServer();
  ```

IsServerKeyProvisioned() method
Check if the public key of the SAP Mobile Server is provisioned on the client.

Syntax
`boolean IsServerKeyProvisioned ()` throws `ODPException`

Returns
'true' if the key is provisioned, else 'false'

Exceptions

- **ODPException class –**

Examples

- **Example 1**

  ```
  if(ODPAppSettings.IsServerKeyProvisioned())
  {
  ...
  }
  ```

ODPCertificateManager class
Consists of methods used to provide the certificate store.

Syntax
`public class  ODPCertificateManager`

ODPCertInfo class
Consists of the headers for X.509 certificate which can be displayed on the certificate selection screen.

Syntax
`public class  ODPCertInfo`

ODPCertInfo(CertInfo) constructor

Syntax

`ODPCertInfo ( CertInfo` *certInfo* `)`

getCertificateDisplayName() method

Display the name of the certificate.

Syntax

`String getCertificateDisplayName ()`

Returns

Certificate names list

Examples

- **Example 1**

```
String s =
ODPCertifcateManager.ODPCertInfo.getCertificateDisplayName();
```

getIssuer() method

Issue the certificate.

Syntax

`String getIssuer ()`

Returns

Distinguish format name for certificate issue authority

Examples

- **Example 1**

```
String s = ODPCertifcateManager.ODPCertInfo.getIssuer();
```

getIssuerCN() method

Retrieve the common name for certificate issue authority.

Syntax

`String getIssuerCN ()`

Returns

Common name of the certificate issue authority

Examples

- **Example 1**

  ```
  String s = ODPCertifcateManager.ODPCertInfo.getIssuerCN();
  ```

getSubject() method

Retrieves the subject of certificate.

Syntax

`String getSubject ()`

Returns

Distinguish format name for subject of the certificate

Examples

- **Example 1**

  ```
  String s = ODPCertifcateManager.ODPCertInfo.getSubject();
  ```

getSubjectCN() method

Retrieve the common name for the certificate.

Syntax

`String getSubjectCN ()`

Returns

Common name of the certificate

Examples

- **Example 1**

  ```
  String s = ODPCertifcateManager.ODPCertInfo.getSubjectCN();
  ```

getValidityBeginDate() method

Retrieve the validity start date of the certificate.

Syntax

`Date getValidityBeginDate ()`

Returns

Validity start date

Examples

- **Example 1**

```
Date d = ODPCertifcateManager.ODPCertInfo.getValidityBeginDate();
```

getValidityExpiryDate() method

Retrieve the validity expiry date of the certificate.

Syntax

`Date getValidityExpiryDate ()`

Returns

Validity end date

Examples

- **Example 1**

```
Date d =
ODPCertifcateManager.ODPCertInfo.getValidityExpiryDate();
```

getSignedCertificateFromStore(ODPCertInfo) method

Retrieve the requested certificate from the BlackBerry device key store as a base64 encoded string.

Syntax

`String getSignedCertificateFromStore ( ODPCertInfo` *certInfo* `)` throws `ODPException,IOException`

Parameters

- **certInfo –** ODPCertInfo object of the desired certificate.

Returns

Certificate

Exceptions

- **ODPException class –**
- **IOException –**

Examples

- **Example 1**

```
String s =
ODPCertificateManager.getSignedCertificateFromStore(odpcertinfo);
```

listAvailableCertificatesFromStore() method
Retrieve a list of ODPCertInfo structures that represent available certificates in the Blackberry device Keystore.

Syntax
`Vector listAvailableCertificatesFromStore ()` throws `ODPException`

Returns
List of ODPCertInfo structures that represents the available certificates in the Keystore

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
Vector v =
ODPCertifcateManager.listAvailableCertificatesFromStore();
```

Usage

User can create pick list for desired certificate.

ODPClientConnection class
Consists of the methods used for client-server connection.

Syntax
`public class  ODPClientConnection`

addConfigurationChangeListener(IODPConfigurationChangeListener) method
Set the listener object that is notified with change in application end-point or push end-point.

Syntax
`void addConfigurationChangeListener`
`( IODPConfigurationChangeListener` *configListener* `)` throws
`ODPException`

Parameters

- **configListener –** Object implements the IODPConfigurationChangeListener interface to invoke the change in application end-point or push end-point.

Examples

- **Example 1**

```
IODPConfigurationChangeListener configListener;
ODPClientConnection.initInstance();
ODPClientConnection occ=ODPClientConnection.getInstance();
occ.addConfigurationChangeListener(configListener);
```

clearServerVerificationKey() method

Invoke this method before registering the new user when the device is connected to new the SAP Mobile Server.

Syntax

`void clearServerVerificationKey ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection.clearServerVerificationKey();
```

getInstance() method

Retrieve an instance of ODPClientConnection class.

Syntax

`ODPClientConnection  getInstance ()` throws `ODPException`

Returns

ODPClientConnectionClass instance

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection.initInstance();
ODPClientConnection occ=ODPClientConnection.getInstance();
```

initInstance(String) method

Initialize the ODPClientConnection.

Syntax

`void initInstance ( String` *appID* `)` throws `ODPException`

Parameters

- **appID –** Application name.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPClientConnection.initInstance();
```

registerForNativePush(IODPPushNotificationListener) method

Register the native push listener.

Syntax

`void registerForNativePush ( IODPPushNotificationListener` *listener* `)` throws `ODPException`

Parameters

- **listener –** Native push listener.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
registerForNativePush(iodppushnotificationlistener);
```

registerForPayloadPush(ISDMNetListener) method

Register the payload push listener.

Syntax

`void registerForPayloadPush ( ISDMNetListener` *listener* `)`

Parameters

- **ISDMNetlistener –** Payload push listener.

Examples

- **Example 1**

  ```
  ODPClientConnection.registerForPayloadPush(isdmnetListener);
  ```

setODPHTTPAuthChallengeListener(IODPHTTPAuthChallengeListener) method

Register a listener for HTTP authentication challenge callback.

Syntax

`void setODPHTTPAuthChallengeListener`
`( IODPHTTPAuthChallengeListener` *listener* `)` throws `ODPException`

Parameters

- **listener –** Listener object that implements ODPClientListeners.IODPHTTPAuthChallengeListener whose callback method is called whenever the connection receives an HTTP_AUTH_FAILURE (HTTP response code 401) challenge.

Exceptions

- **ODPClientConnectionException –**

Examples

- **Example 1**

  ```
  IODPHTTPAuthChallengeListener listener;
  ODPClientConnection.setODPHTTPAuthChallengeListener(listener);
  ```

setODPHttpErrorListener(IODPHttpErrorListener) method

Ensure that OData BlackBerry clients are notified of HTTP errors while establishing a connection with the network edge.

Syntax

`void setODPHttpErrorListener ( IODPHttpErrorListener` *listener* `)` throws `ODPException`

Parameters

- **listener –** Listener object that implements the interfaceIODPHttpErrorListener.

Exceptions

- **MessagingClientException –**

Examples

- **Example 1**

```
public class UserRegistration implements
IODPHttpErrorListener{ public void startUserRegistration(){
UserManager.initialize(appID);
UserManager.setConnectionProfile(serverIP,serverPort,farmID);
UserManager.setODPHttpErrorListener(this);
UserManager.registerUser(username,securityConfig,password);
}
//callback method for HttpError
public void onHttpError(int errorCode, String errorMsg,
Hashtable errorHeader) {
logger.info(null, "On HttpError", "Error Info" +errorCode
+errorMsg);
}
```

setPushListener(ISDMNetListener) method

Consume push messages, the application registers a listener object.

Syntax

`void setPushListener ( ISDMNetListener` *pushListener* `)`

Parameters

- **pushListener –** Object that implements the ISDMNetListener interface is be invoked whenever there is a push message from the server.

Examples

- **Example 1**

```
Listener Object
UserManager.setPushListener(listenerObjectFromApp);
Implementation of APIs in the Listener Object
ISDMNetListener pushListener;
ODPClientConnection.initInstance();
ODPClientConnection occ=ODPClientConnection.getInstance();
occ.setPushListener(pushListener);
```

ODPClientListeners class

Consists of the methods used by application to get notification from various events and respond to them.

Syntax

```
public class  ODPClientListeners
```

ODPHTTPAuthChallengeCredentials class

Contains credentials that is returned from getCredentials callback method on an HTTP 401 challenge.

Syntax

```
public class  ODPHTTPAuthChallengeCredentials
```

ODPHTTPAuthChallengeCredentials(String, String) constructor

Syntax

`ODPHTTPAuthChallengeCredentials ( String` *username* `,  String` *password* `)`

IODPConfigurationChangeListener interface

Register to notify the changes in application end-point and push end-point.

Syntax

```
public interface  IODPConfigurationChangeListener
```

onConfigurationChange(int, String) method

Send notification if there is any change in push or application end-point.

Syntax

`void onConfigurationChange ( int` *key* `,  String` *newValue* `)`

Parameters

- **key –** Key of the changed end-point.
- **newValue –** New end-point.

Examples

- **Example 1**

```
public class ApplicationClass implements
IODPConfigurationChangeListener{
public void onConfigurationChange(int PropertyID, String
Property){
// some application related processing
}
```

CUSTOMIZATION_RESOURCES variable

Syntax
`int CUSTOMIZATION_RESOURCES`

PROXY_APPLICATION_ENDPOINT variable

Syntax
`int PROXY_APPLICATION_ENDPOINT`

PROXY_PUSH_ENDPOINT variable

Syntax
`int PROXY_PUSH_ENDPOINT`

PWDPOLICY_CHANGED variable

Syntax
`int PWDPOLICY_CHANGED`

PWDPOLICY_DEFAULT_PASSWORD_ALLOWED variable

Syntax
`int PWDPOLICY_DEFAULT_PASSWORD_ALLOWED`

PWDPOLICY_ENABLED variable

Syntax
`int PWDPOLICY_ENABLED`

PWDPOLICY_EXPIRES_IN_N_DAYS variable

Syntax
```
int PWDPOLICY_EXPIRES_IN_N_DAYS
```

PWDPOLICY_HAS_DIGITS variable

Syntax
```
int PWDPOLICY_HAS_DIGITS
```

PWDPOLICY_HAS_LOWER variable

Syntax
```
int PWDPOLICY_HAS_LOWER
```

PWDPOLICY_HAS_SPECIAL variable

Syntax
```
int PWDPOLICY_HAS_SPECIAL
```

PWDPOLICY_HAS_UPPER variable

Syntax
```
int PWDPOLICY_HAS_UPPER
```

PWDPOLICY_LENGTH variable

Syntax
```
int PWDPOLICY_LENGTH
```

PWDPOLICY_LOCK_TIMEOUT variable

Syntax
```
int PWDPOLICY_LOCK_TIMEOUT
```

PWDPOLICY_MIN_UNIQUE_CHARS variable

Syntax
```
int PWDPOLICY_MIN_UNIQUE_CHARS
```

PWDPOLICY_RETRY_LIMIT variable

Syntax

`int PWDPOLICY_RETRY_LIMIT`

IODPHTTPAuthChallengeListener interface

Implement to listen to HTTP_AUTH_FAILURE(HTTP response code 401) challenge.

Syntax

`public interface  IODPHTTPAuthChallengeListener`

getCredentials(String, String, String) method

Retrieve the credentials on the connection failure.

Syntax

`ODPHTTPAuthChallengeCredentials getCredentials ( String` *sHostName* `,  String` *sUserName* `,  String` *sRealm* `)`

Parameters

- **sHostName –** Server host name.
- **sUserName –** User name which is last used, empty string on the first callback for the current run of the application.
- **sRealm –** Realm specified in the 401 challenge headers received from the server, if available. If the realm is not available in the response, an empty string will be passed.

Returns

ODPHTTPAuthChallengeCredentials structure

Examples

- **Example 1**

```
public class ApplicationClass implements
IODPHTTPAuthChallengeListener{
ODPHTTPAuthChallengeCredentials getCredentials(String host,
String username, String realm){
// some application related processing
return odphttpauthchallengecredentialsobj;
}
```

Usage

This method throws HTTP response code HTTP_AUTH_FAILURE - 401. The application is expected to return a structure from the method containing the credentials to be used. The argument contains the credentials last used, which will be empty strings on the first callback for the current run of the application (credentials are not persisted). Once credentials are received, they are used in subsequent connections until challenged again.

IODPHttpErrorListener interface

Implement the listener to ensure that OData BlackBerry clients are notified of HTTP errors while establishing a connection with the network edge.

Syntax

`public interface IODPHttpErrorListener`

OnHttpError(int, String, Hashtable) method

Call when there is an error occurred during connection with reverse proxy/relay server.

Syntax

`void OnHttpError ( int` *errorCode* `, String` *errorMsg* `, Hashtable` *httpHeader* `)`

Parameters

- **errorCode –** Error code of the message.
- **errorMsg –** Error message.
- **httpHeader –** Response headers with error message.

Examples

- **Example 1**

```
public class ApplicationClass implements IODPHttpErrorListener{
public void onHttpError(int erroCode, String errorMsg, Hashtable
httpHeader){
// some application related processing
}
```

IODPPushNotificationListener interface

The application should register and implement a listener interface to receive native or payload push notifications.

Syntax

`public interface IODPPushNotificationListener`

Remarks

The IODPPushNotificationListener interface should be implemented by the application to receive the BES native push notifications on BlackBerry devices. The onPushNotification method is called when a new BES push notification is received.

Note: When the application is not running or terminated, the BES notifications will not reach the client device. The IODPPushNotificationListener will not be invoked.

onPushNotification(Hashtable) method

Call this method when a new BES push notification is received.

Syntax

`int onPushNotification ( Hashtable` *ipushdata* `)`

Parameters

- **ipushdata –** Hashtable containing the data,part of the notification.

Returns

0 - Reconnect to server and retrieve the payload 1 - Do not reconnect to server to retrieve the payload

Examples

- **Example 1**

```
public class ApplicationClass implements
IODPPushNotificationListener{
public int onPushNotification(Hashtable idata){
// some application related processing
return 0;
}
```

IODPUserRegistrationListener interface

Objects registered with this Interface can be registered to notify the result of user registration.

Syntax

```
public interface  IODPUserRegistrationListener
```

onAsyncRegistrationResult(boolean, int, String) method

Callback method that is invoked with the result of user registration.

Syntax

`void onAsyncRegistrationResult ( boolean` *registrationSuccess* `,  int` *errCode* `,  String` *errMsg* `)`

Parameters

- **registrationSuccess –** Return "true" when user registration is successful , "false" otherwise.
- **errCode –** Error code in case of unsuccessful user registration.
- **errMsg –** Error message shows user registration failure.

Examples

- **Example 1**

  ```
  public class ApplicationClass implements
  IODPUserRegistrationListener{
  public void onAsyncRegistrationResult(boolean
  registrationsuccess, int errorcode, String errmsg){
  // some application related processing
  }
  ```

ODPException class

This Class represents the exception thrown by the client library.

Syntax

`public class ODPException`

ODPException(int) constructor

Constructs a UserManagerException with specified error code.

Syntax

`ODPException ( int` *errorCode* `)`

ODPException(int, String) constructor

Constructs a UserManagerException with specified error code and error message.

Syntax

`ODPException ( int` *errorCode* `, String` *message* `)`

ODPException(String) constructor

Constructs a UserManagerException with specified error message.

Syntax

`ODPException ( String` *message* `)`

getErrorCode() method

Returns the error code of this UserManagerException.

Syntax

`int getErrorCode ()`

ANY_INPUT_FIELD_NULL variable

User registration fields are null.

Syntax

`final int ANY_INPUT_FIELD_NULL`

APPLICATION_ID_NULL variable
Application is not initialized.

Syntax
```
final int APPLICATION_ID_NULL
```

APPLICATION_USER_ALREADY_REGISTERED variable
Application user is already registered.

Syntax
```
final int APPLICATION_USER_ALREADY_REGISTERED
```

APPLICATION_USER_NOT_REGISTERED variable
Application user is unregistered.

Syntax
```
final int APPLICATION_USER_NOT_REGISTERED
```

EMPTY_RESPONSE_FROM_SERVER variable
Empty response received from server.

Syntax
```
final int EMPTY_RESPONSE_FROM_SERVER
```

INTERNAL_PARSING_ERROR variable
SAP Mobile Server internal parsing error.

Syntax
```
final int INTERNAL_PARSING_ERROR
```

INVALID_MODULE_ID variable
Invalid module handle ID.

Syntax
```
final int INVALID_MODULE_ID
```

JSON_PARSING_FAILED variable
Internal error, JSON parsing has failed.

Syntax
```
final int JSON_PARSING_FAILED
```

REGISTRATION_FAILED_UNKNOWN_ERROR variable
User registration timed out.

Syntax
`final int REGISTRATION_FAILED_UNKNOWN_ERROR`

REGISTRATION_LISTENER_NULL variable
Asynchronous user registration listener is unregistered with UserManager.

Syntax
`final int REGISTRATION_LISTENER_NULL`

SINGLETON_INITIALIZATION_FAILED variable
Singleton initialization failed.

Syntax
`final int SINGLETON_INITIALIZATION_FAILED`

ODPUserManager class
Single entry point for all user on-boarding interactions.

Syntax
`public class ODPUserManager`

Remarks
It exposes methods to manage the user credentials.

cleanUp() method
Clear the persistent storage.

Syntax
`void cleanUp ()`

Examples

- **Example 1**

```
ODPUserManager.cleanup();
```

deleteUser() method
Deletes a registered application connection.

Syntax
`void deleteUser ()` throws `ODPException`

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpuserRegistrationManager.deleteUser();
```

enableHTTPS(boolean) method

Set the HTTPS as the transport protocol that the OData client should use to communicatewith any host.

Syntax

`void enableHTTPS ( boolean` *useHTTPS* `)` throws `ODPException`

Parameters

- **useHTTPS –** Set to "true", if the protocol to be used is HTTPS. Set to "false", if the protocol to be used is HTTP. Note: The default protocol is HTTP.

Exceptions

- **ODPClientConnectionException –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpuserRegistrationManager.enableHTTPS(useHTTPS);
```

Usage

For example, Relay server.

getInstance() method

Return an object of ODPUserManager.

Syntax

`ODPUserManager  getInstance ()` throws `ODPException`

Returns
ODPUserManager object

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.initInstance();
ODPUserManager oum=ODPUserManager.getInstance();
```

initInstance(String) method
Initialize the ODPUserManager.

Syntax
`void initInstance ( String` *appID* `)` throws `ODPException`

Parameters

- **appID –** Application identifier.

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(appid);
```

Usage

Before you use any of the other BlackBerry ODP APIs, you have to first initialize an application.

isUserRegistered() method
Check if a device user is registered or not.

Syntax
`boolean isUserRegistered ()`

Returns
True - if the user is already registered False - if the user is not registered

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(appid);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
if(!odpuserRegistrationmanager.isUserRegistered())
{
...
}
```

setConnectionProfile(String, int, String) method

Set the connectivity details of the SAP Mobile Server.

Syntax

`void setConnectionProfile ( String` *host* `,  int` *port* `,  String` *farmID* `)` throws `ODPException`

Parameters

- **host –** IP Address of the ODP server.
- **port –** Port number of the server.
- **farmID –** farm ID of the SAP Mobile Server

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpuserRegistrationmanager.setConnectionProfile(host, port,
farmID);
```

Usage

Typically, this information comes as an input from the application via a 'Settings' screen.

setConnectionProfileFromFile(String) method

Connection settings for an application can be provisioned by providing a file location.

Syntax

`void setConnectionProfileFromFile ( String` *filePath* `)` throws `ODPException`

Parameters

- **filePath –** : location of file on device, containing connection details, such as host, post, farm-id.

Exceptions

- **MessagingClientException –**

Examples

- **Example 1**

```
ODPUserManager.setConnectionProfileFromFile("data/app/
Sybase_Messaging_com.sap.android.trialapp-1.cfg");
```

setHttpHeaders(Hashtable, Hashtable) method

Enable the users to set the HTTP headers and cookies.

Syntax

`void setHttpHeaders ( Hashtable` *param1* `, Hashtable` *param2* `)` throws `ODPException`

Parameters

- **param1 –** - HTTP header
- **param2 –** - Cookies

Exceptions

- **ODPException class –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpuserRegistrationManager.setHttpHeaders(param1,param2);
```

setRelayServerURLTemplate(String) method

Invoke this method to set the property "ADVANCED_RELAY_SVR_URL_TEMPLATE".

Syntax

`void setRelayServerURLTemplate ( String` *URLSuffix* `)` throws `ODPException`

Parameters

- **URLSuffix –** The value for property ADVANCED_RELAY_SVR_URL_TEMPLATE.

Exceptions

- **ODPException class –**
- **MessagingClientException –**

Examples

- **Example 1**

```
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpuserRegistrationManager.setRelayServerURLTemplate(URLSuffix);
```

setUserRegistrationListener(ODPClientListeners.IODPUserRegistrationListener) method

Set the listener object that is notified with the result of user registration.

Syntax

`void setUserRegistrationListener`
`( ODPClientListeners.IODPUserRegistrationListener`
registrationListener `)`

Parameters

- **registrationListener –** Object implements the IUserRegistrationListener interface and is invoked with the result of user registration.

Examples

- **Example 1**

```
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpuserRegistrationManager.setUserRegistrationListener(registrati
onListener);
```

CHAPTER 5 Using a Reverse Proxy for OData Applications

Requirements for third party reverse proxies for use with SAP Mobile Platform.

The reverse proxy must be a straight pass-though proxy server. Ensure the reverse proxy meets these requirements:

- Does not change the content encoding of the requests or responses. Chunked transfer encoding is the required data transfer mechanism. Content-length encoding is not supported.
- Does not remove any HTTP headers.
- The timeout period (if any) must be greater than the timeout used by the clients.
- The resulting URL that is passed to the SAP Mobile Platform must be `http://HostName:port`.

The OData application needs to communicate to the message server port. The configuration policies are:

1. Map the root context of `http://reverseProxy:5001` to `http://supServer:5001`. Set connection properties just as you would to directly connect to SAP Mobile Server.
2. Map the "/smp/message" context of `http://reverseProxy:8080` to `http://supServer:5001`. The OData application needs to add the "/smp/message" as URL suffix for connection properties during registration.

CHAPTER 6 Using REST API Services with Online Data Proxy

Client applications can make use of native HTTPS services to connect to the server.

The HTTP(S) channel supports a reverse proxy that is used to forward the request from the client in a public network to the SAP Mobile Server located inside a corporate network.

The OData SDK provides functionality required for parsing, caching and persistency to the application. Every time an application has to fetch data from the back-end system, it can make use of native HTTP(s) APIs with which it can connect to the HTTP channel. Using specific request header formats to establish the connection.

For more information on using this channel, see *Developer Guide: REST API Applications*.

Index