OData App Development using Messaging Channel

SAP Mobile Platform 3.0 SP03
Using HTTPS over the SAP Mobile Platform Messaging Channel .................................... 14
  Enabling HTTPS as a Transport Protocol ........... 15
  Verifying a Server Certificate ........................... 15
  Enabling a Listener for HTTPS Support with
    Basic Authentication Challenge ....................... 15
  Registering a Listener for HTTP Error in
    BlackBerry Applications ................................... 16
Debugging Runtime Error and Performance Analysis
  Analyzing Performance Data Points .................. 17
Deploying Applications to Devices ...................... 19
  Signing ......................................................... 19
  Provisioning Options for BlackBerry Devices ....... 19
  BES Provisioning for BlackBerry ....................... 20
  BlackBerry Desktop Manager Provisioning .......... 20
OData SDK Components and APIs .......................... 21
  SDMParser ..................................................... 22
    ETag Support ................................................ 27
    Examples ..................................................... 27
  SDMCache ..................................................... 28
    SDMCache Public APIs .................................... 28
  SDMPersistence ................................................. 29
    SDMPersistence Public APIs ............................ 29
  SDMConnectivity .............................................. 33
    SDMConnectivity Public APIs .......................... 33
  SDMRequest ................................................... 35
  SDMResponse .................................................. 37
  ETag Support .................................................. 38
  SDMConfiguration .............................................. 39
    SDMConfiguration Public APIs ......................... 39
  SDMSupportability ............................................ 40
    SDMLogger .................................................. 40
    SAP Passport ............................................... 41
ODP SDK API Usage ............................................. 41
Contents

Security APIs .................................................................................42
DataVault .........................................................................................42
ODP SDK API Reference for BlackBerry ........................................60
client package .............................................................................60

CHAPTER 3: Using a Reverse Proxy for OData
Applications .....................................................................................89

CHAPTER 4: Using REST API Services with Online
Data Proxy ........................................................................................91

Index ........................................................................................................93
CHAPTER 1  Overview

This guide describes how to build applications for BlackBerry devices using the OData 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/HTTPS 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 the BlackBerry platform and it is based on the native device SDK of the platform. Native applications installed on BlackBerry
devices allow the client application to leverage the support provided by the 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 a detailed platform specific description, see the respective chapter on BlackBerry. For detailed platform specific descriptions, see the respective chapters on Android, BlackBerry and iOS.

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

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&c=firsttab&a=0&p=categories, then navigate to the most current version.
CHAPTER 1: Overview
CHAPTER 2 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 9

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.
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 S:\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 the SUP server synchronously 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 Platform Messaging 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 5
• Deploying Applications to Devices on page 19

**Initializing the Application**

Initialize the application when it starts the first time.

The following illustrates how to initialize an application.

```java
ODPUserManager.initInstance("Testing");
ODPUserManager userApp = ODPUserManager.getInstance();
```

**See also**

• Setting Server Details on page 11

**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.

```java
Public class Mylistener implements IODPPushNotificationListner
{
    Public int onPushNotification(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

```java
public static void setPushListener(com.sap.mobile.lib.sdmconnectivity.ISDBNetListener pushListener)
```

Parameters

- **pushListener** – Object that implements ISDMNetListener interface.

Examples

- **Listener Object**
  ```java
  UserManager.setPushListener(listenerObjectFromApp);
  ```

- **Implementation of APIs in the Listener Object**
  ```java
  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.

```java
userApp.setConnectionProfile("155.56.51.245", 123, "test.farmMBS");
```

See also

- *Initializing the Application* on page 10

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.

```java
userApp.registerUser("perfios", "SSO", "perfios", true);
```
**Sending Data Request to the Back-end**

You can forward data request messages to the back-end through the SUP server synchronously 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 14

**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.

```java
ISDMConnectivityParameters 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");
parameters.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 up to which the application waits until the request reaches the server.

The following code illustrates how to set the required timeout.

```java
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.

```java
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.

```java
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 12

Using HTTPS over the SAP Mobile Platform Messaging 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 17
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.IODPHTTPAuthChallengeListener 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 –
  ```java
  ODPUserManager.enableHTTPS(true);
  ODPUserManager.setODPHTTPAuthChallengeListener(this);
  ```

- Implementation of the Listener
  ```java
  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
  ```java
  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

```java
public static void setODPHTTPErrorResponseListener(IODPHttpErrorResponseListener oListener) throws MessagingClientException
```
Parameters

- **oListener** – listener object that implements the interface `IODPHttpErrorListener`.

Examples

- **Implement the listener**

  ```java
  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.

```java
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 Platform Messaging Channel* on page 14

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.

<table>
<thead>
<tr>
<th>Log Reading</th>
<th>Application</th>
<th>Proxy Client</th>
<th>Messaging Client</th>
<th>Network (includes reverse proxy/relay server)</th>
<th>SAP Mobile Platform Server</th>
<th>Network</th>
<th>Enterprise Information System (EIS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2E: RR</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ODP: RR</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IMO: RR</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Network: RR</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

**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 9
- OData SDK Components and APIs on page 21

**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:


**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.

<table>
<thead>
<tr>
<th>Provisioning Method</th>
<th>Purpose</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackBerry Enterprise Server (BES) Over-the-Air (OTA)</td>
<td>Enterprise installations</td>
<td>When the BlackBerry device activates, it automatically pairs with the BES and downloads the application. See <a href="http://www.blackberry.com/btsc/search.do?cmd=displayKC&amp;docType=kc&amp;externalId=KB03748">http://www.blackberry.com/btsc/search.do?cmd=displayKC&amp;docType=kc&amp;externalId=KB03748</a> for step-by-step instructions.</td>
</tr>
<tr>
<td>OTA: URL/link to installation files</td>
<td>Enterprise installations</td>
<td>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.</td>
</tr>
<tr>
<td>Desktop Manager</td>
<td>Personal installation</td>
<td>Installs the application when the BlackBerry device is synced via a computer.</td>
</tr>
</tbody>
</table>

**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 ISDMParseCache interface from SDMCommon.

**See also**
- *Deploying Applications to Devices* on page 19
- *ODP SDK API Usage* on page 41

**SDMParse**

The parser (SDMParse 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:

```java
/**
 * 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 SDMParseException If the XML source is invalid.
 * @throws IllegalArgumentException If the argument is null.
 */
public abstract ISDMParserDocument parseXML(byte[] xml) throws SDMParseException, 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:

```java
/**
 * 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
 */
```
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 */
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 SDM0Data Service Document XML and converts it to an appropriate Java Object.
 * @param xml
 * @return ISDM0DataServiceDocument The Object representation of SDM0Data Service Document.
 * @throws SDMParseException
 * @throws IllegalArgumentException
 * @throws NullPointerException
*/
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.
 */
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.

```java
/**
* 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 SDMParseException
* @throws IllegalArgumentException
*/
public abstract ISDMODataFeed parseSDMODataEntriesXML(byte[] xml, ISDMODataEntitySet eSet) throws SDMParseException, IllegalArgumentException;
```

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 ISMDOMetadata 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.
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

```
private SDMDataEntry 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 setSDMDataServiceDocument(ISDMDataServiceDocument serviceDocument);
void setSDMDataMetadata(ISDMDataMetadata metadata);
void setSDMDataEntries(Vector entries, String collectionId);
ISDMDataServiceDocument getSDMDataServiceDocument();
ISDMDataSchema getSDMDataSchema();
ISDMDataMetadata getSDMDataMetadata();
ISDMDataEntry getSDMDataEntry(String key);
Vector getSDMDataEntries(String collectionId);
Vector getStoredDocuments();
Vector searchSDMDataEntries(String searchTerm, String collectionId);
void removeSDMDataServiceDocument();
void removeSDMDataSchema();
void removeSDMDataEntry(String key);
void removeSDMDataEntries(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**

<table>
<thead>
<tr>
<th>ISDMPersistence</th>
</tr>
</thead>
<tbody>
<tr>
<td>void storeCache(final ISDMCache cache)</td>
</tr>
<tr>
<td>void storeCacheSecured(final ISDMCache cache)</td>
</tr>
<tr>
<td>void storePreferencesSecured(final ISDMPreferences preferences)</td>
</tr>
<tr>
<td>ISDMCache loadCache(ISDMCache cache, ISDMParser parser)</td>
</tr>
<tr>
<td>ISDMCache loadCacheSecured(ISDMCache cache, ISDMParser parser)</td>
</tr>
<tr>
<td>void loadPreferencesSecured(final ISDMPreferences preferences)</td>
</tr>
<tr>
<td>void storeObject(final long key, final Object object)</td>
</tr>
<tr>
<td>Object loadObject(final long key)</td>
</tr>
<tr>
<td>void clearCache()</td>
</tr>
<tr>
<td>void clearObject(final long key)</td>
</tr>
</tbody>
</table>

**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:
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:
The main method of the application has to be extended with a branch for the background process, which registers itself for code module changes:

```java
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.
    }
}
```
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.

```java
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(SDMConstants.SERVICE_DOC_KEY));
            PersistentStore.destroyPersistentObject(constants.getId(SDMConstants.METADATA_KEY));
            PersistentStore.destroyPersistentObject(constants.getId(SDMConstants.DATA_ENTRY_KEY));
            PersistentStore.destroyPersistentObject(constants.getId(SDMConstants.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 code.
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

```java
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:

```java
ISDMRequestManager

void makeRequest(final ISDMRequest aRequest);
void makeRequest(final ISDMBundleRequest aBundleRequest)
ISDMConnectivityParameters 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:

```java
ISDMConnectivityParameters

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

```java
//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
reqManager.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.

```java
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:

```java
ISDMRequest

void setRequestUrl(final String aUrl)
String getRequestUrl()
void setRequestMethod(final int aRequestMethod)
int getRequestMethod()
```
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:

```java
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:

```java
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

```java
public void onSuccess(ISDMRequest aRequest, SDMHttpResponse aResponse) {
    System.out.println("Http response status code:" + aResponse.getStatus()
```
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 Android
        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 operating parameters of an HTTP transaction. The following header fields are applicable for ETag support.

**Table 1. ETag Header Fields**

<table>
<thead>
<tr>
<th>Header Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>If-Match</td>
<td>A client that has one or more ETags obtained from a resource, can verify that one of these ETags is current.</td>
</tr>
<tr>
<td>If-None-Match</td>
<td>A client that has one or more ETags obtained from a resource can verify that none of these ETags is current.</td>
</tr>
<tr>
<td>If-Range</td>
<td>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.</td>
</tr>
</tbody>
</table>

*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**

```java
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 getLogRecordsByComponentName(final String componentName)
Vector getLogRecordsByTimeStamp(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`
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

```java
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`.

```java
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

```java
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.

```java
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.

```java
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**

```java
public abstract DataVault.DVDataName[] getDataNames()
```

**Parameters**

None.

**Returns**

Returns a DVPasswordPolicy object, as an array of DVDataName structure objects.

Examples

- **Get data names**

```java
// 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**

```java
public abstract void setPasswordPolicy(DataVault.DVPasswordPolicy oPasswordPolicy)
```

**Parameters**

- `oPasswordPolicy` – the password policy constraints.

**Returns**
None.

**Examples**

- **Set a password policy**

  ```java
  // 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>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultPasswordAllowed</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>minimumLength</td>
<td>Integer</td>
<td>The minimum length of the password.</td>
</tr>
<tr>
<td>hasDigits</td>
<td>Boolean</td>
<td>Indicates if the password must contain digits.</td>
</tr>
<tr>
<td>hasUpper</td>
<td>Boolean</td>
<td>Indicates if the password must contain uppercase characters.</td>
</tr>
<tr>
<td>hasLower</td>
<td>Boolean</td>
<td>Indicates if the password must contain lowercase characters.</td>
</tr>
<tr>
<td>hasSpecial</td>
<td>Boolean</td>
<td>Indicates if the password must contain special characters. The set of special characters is: “~!@#$%^&amp;*()-+”.</td>
</tr>
<tr>
<td>expirationDays</td>
<td>Integer</td>
<td>Specifies password expiry days from the date of setting the password. 0 indicates no expiry.</td>
</tr>
<tr>
<td>minUniqueChars</td>
<td>Integer</td>
<td>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, “aaroa” would be a valid password.</td>
</tr>
<tr>
<td>lockTimeout</td>
<td>Integer</td>
<td>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.</td>
</tr>
<tr>
<td>retryLimit</td>
<td>Integer</td>
<td>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.</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD_REQUIRED</td>
<td>50</td>
<td>Indicates that a blank or null password was used when the password policy does not allow default password.</td>
</tr>
<tr>
<td>PASSWORD_UNDER_MIN_LENGTH</td>
<td>51</td>
<td>Indicates that the password length is less than the required minimum.</td>
</tr>
<tr>
<td>PASSWORD_REQUIRES_DIGIT</td>
<td>52</td>
<td>Indicates that the password does not contain digits.</td>
</tr>
<tr>
<td>PASSWORD_REQUIRES_UPPER</td>
<td>53</td>
<td>Indicates that the password does not contain upper case characters.</td>
</tr>
<tr>
<td>PASSWORD_REQUIRES_LOWER</td>
<td>54</td>
<td>Indicates that the password does not contain lower case characters.</td>
</tr>
<tr>
<td>PASSWORD_REQUIRES_SPECIAL</td>
<td>55</td>
<td>Indicates that the password does not contain one of these special characters: <code>~!#$%^&amp;*()-+.</code></td>
</tr>
<tr>
<td>PASSWORD_UNDER_MIN_UNIQUE</td>
<td>56</td>
<td>Indicates that the password contains fewer than the minimum required number of unique characters.</td>
</tr>
<tr>
<td>PASSWORD_EXPIRED</td>
<td>57</td>
<td>Indicates that the password has been in use longer than the number of configured expiration days.</td>
</tr>
</tbody>
</table>

**getPasswordPolicy android blackberry**
Retrieves the password policy set by `setPasswordPolicy`.

Use this method once the DataVault is unlocked.
**Syntax**

```java
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**

  ```java
  // 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**

```java
public boolean isDefaultPasswordUsed()
```

**Parameters**

None.

**Returns**

<table>
<thead>
<tr>
<th>Returns</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Both the default password and the default salt are used to encrypt the vault.</td>
</tr>
<tr>
<td>false</td>
<td>Either the default password or the default salt is not used to encrypt the vault.</td>
</tr>
</tbody>
</table>

**Examples**

- **Check if default password used**

  ```java
  // 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 vault’s properties or stored content.
  vault.lock();

isLocked
Checks whether the vault is locked.

Syntax
public boolean isLocked()

Parameters
None.

Returns

<table>
<thead>
<tr>
<th>Returns</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The vault is locked.</td>
</tr>
<tr>
<td>false</td>
<td>The vault is unlocked.</td>
</tr>
</tbody>
</table>
**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**

```java
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.
  ```java
  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**

```java
public void setString(
    String name,
```
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.

```java
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

```java
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.

```java
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**

```java
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.
```java
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**  
```java
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.

```java
try {
    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.
  ```java
  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**

  ```java
  // 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.

```java
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.

```java
    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[1!A" );

    // Call isDefaultPasswordUsed() and we will see that the default
    password is NOT used anymore.
    isDefaultPasswordUsed = oDataVault.isDefaultPasswordUsed();
}
```

```java
    catch( Exception exception )
    {
    }
```

```java
    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**
  ```java
  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

```java
Object o = getProperty(key);
```

*setProperty(String, Object) method*
Store the application properties as key value pairs.

**Syntax**
```java
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

```java
AppSettingsStore.setProperty(key, value);
```

*ODPAppSettings class*
Consists of methods used to retrieve the setting details required by the application.

**Syntax**
```java
public class ODPAppSettings
```

*getApplicationEndPoint() method*
Retrieve the application end-point with which you can access the business data.

**Syntax**
```java
String getApplicationEndPoint() throws ODPException
```

**Returns**
Application end-point

**Exceptions**
- ODPException. –
Examples

- Example 1

```java
String endpoint = ODPAppSettings.getApplicationEndPoint();
```

**getFarmID() method**
Retrieve the farm ID provisioned in the client repository.

**Syntax**
```java
String getFarmID() throws ODPException
```

**Returns**
Farm ID

**Exceptions**
- ODPException class –

Examples

- Example 1

```java
String fg = ODPAppSettings.getFarmID();
```

**getPasswordPolicy() method**
Retrieve the data vault password policy.

**Syntax**
```java
DVPasswordPolicy getPasswordPolicy() throws ODPException
```

**Returns**
Password policy

**Exceptions**
- ODPException class –

Examples

- Example 1

```java
DVPasswordPolicy dvp = ODPAppSettings.getPasswordPolicy();
```
**getPortNumber() method**
Retrieve the port number provisioned in the client repository.

**Syntax**
```java
int getPortNumber() throws ODPException
```

**Returns**
Port number

**Exceptions**
- ODPException class –

**Examples**
- Example 1
  ```java
  int g=ODPAppSettings.getPortNumber();
  ```

**getPushEndPoint() method**
Retrieve the push end-point which the EIS can use to push data to the ODP client.

**Syntax**
```java
String getPushEndPoint() throws ODPException
```

**Returns**
Push end-point

**Exceptions**
- ODPException class –

**Examples**
- Example 1
  ```java
  String pushendpoint=ODPAppSettings.getPushEndPoint();
  ```

**getServer() method**
Retrieve the SAP Mobile Server name provisioned in the client repository.

**Syntax**
```java
String getServer() throws ODPException
```

...
**Returns**
SAP Mobile Server name

**Exceptions**
- ODPEXception class –

**Examples**
- Example 1
  ```java
  String server=ODPAppSettings.getServer();
  ```

**IsServerKeyProvisioned() method**
Check if the public key of the SAP Mobile Server is provisioned on the client.

**Syntax**
```java
boolean IsServerKeyProvisioned () throws ODPEXception
```

**Returns**
'true' if the key is provisioned, else 'false'

**Exceptions**
- ODPEXception class –

**Examples**
- Example 1
  ```java
  if(ODPAppSettings.IsServerKeyProvisioned())
  {
  ... 
  }
  ```

**ODPCertificateManager class**
Consists of methods used to provide the certificate store.

**Syntax**
```java
public class ODPCertificateManager
```

**ODPCertInfo class**
Consists of the headers for X.509 certificate which can be displayed on the certificate selection screen.

**Syntax**
```java
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**

  ```java
  String s = ODPCertificateManager.ODPCertInfo.getCertificateDisplayName();
  ```

**getIssuer() method**
Issue the certificate.

**Syntax**

```
String getIssuer()
```

**Returns**
Distinguish format name for certificate issue authority

**Examples**

- **Example 1**

  ```java
  String s = ODPCertificateManager.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

```java
String s = ODPCertificateManager.ODPCertInfo.getIssuerCN();
```

**getSubject() method**

Retrieves the subject of certificate.

**Syntax**

```java
String getSubject()
```

**Returns**

Distinguish format name for subject of the certificate

Examples

- Example 1

```java
String s = ODPCertificateManager.ODPCertInfo.getSubject();
```

**getSubjectCN() method**

Retrieve the common name for the certificate.

**Syntax**

```java
String getSubjectCN()
```

**Returns**

Common name of the certificate

Examples

- Example 1

```java
String s = ODPCertificateManager.ODPCertInfo.getSubjectCN();
```

**getValidityBeginDate() method**

Retrieve the validity start date of the certificate.

**Syntax**

```java
Date getValidityBeginDate()
```

**Returns**

Validity start date
Examples

- Example 1

```java
Date d = ODPCerticateManager.ODPCertInfo.getValidityBeginDate();
```

**getValidityExpiryDate() method**
Retrieve the validity expiry date of the certificate.

**Syntax**

```java
Date getValidityExpiryDate()
```

**Returns**

Validity end date

Examples

- Example 1

```java
Date d = ODPCerticateManager.ODPCertInfo.getValidityExpiryDate();
```

**getSignedCertificateFromStore(ODPCertInfo) method**
Retrieve the requested certificate from the BlackBerry device key store as a base64 encoded string.

**Syntax**

```java
String getSignedCertificateFromStore(ODPCertInfo certInfo) throws ODPException, IOException
```

**Parameters**

- `certInfo` – ODPCertInfo object of the desired certificate.

**Returns**

Certificate

**Exceptions**

- ODPException class –
- IOException –
Examples

• Example 1

```java
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

```java
Vector v = ODPCertificateManager.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**

```java
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
  ```java
  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

```java
void clearServerVerificationKey () throws ODPException
```

Exceptions

- ODPException class –

Examples

- Example 1
  ```java
  ODPClientConnection.clearServerVerificationKey();
  ```

`getInstance()` method

Retrieve an instance of ODPClientConnection class.

Syntax

```java
ODPClientConnection  getInstance () throws ODPException
```

Returns

ODPClientConnectionClass instance

Exceptions

- ODPException class –
CHAPTER 2: Developing BlackBerry Applications

**Examples**

- **Example 1**
  ```java
  ODPClientConnection.initInstance();
  ODPClientConnection occ=ODPClientConnection.getInstance();
  ```

*initInstance(String) method*
Initialize the ODPClientConnection.

**Syntax**
```java
void initInstance ( String appID ) throws ODPException
```

**Parameters**
- **appID** – Application name.

**Exceptions**
- **ODPException class** –

**Examples**

- **Example 1**
  ```java
  ODPClientConnection.initInstance();
  ```

*registerForNativePush(IODPPushNotificationListener) method*
Register the native push listener.

**Syntax**
```java
void registerForNativePush ( IODPPushNotificationListener listener ) throws ODPException
```

**Parameters**
- **listener** – Native push listener.

**Exceptions**
- **ODPException class** –

**Examples**

- **Example 1**
  ```java
  registerForNativePush(iodppushnotificationlistener);
  ```
**registerForPayloadPush(ISDMNetListener) method**
Register the payload push listener.

**Syntax**
void registerForPayloadPush ( ISDMNetListener listener )

**Parameters**
- ISDMNetListener – Payload push listener.

**Examples**
- Example 1
  
  ```java
  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
  
  ```java
  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**
```java
void setODPHttpErrorListener ( IODPHttpErrorListener listener ) throws ODPException
```

**Parameters**
- **listener** – Listener object that implements the interface IODPHttpErrorListener.

**Exceptions**
- **MessagingClientException** –

**Examples**
- **Example 1**
```java
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**
```java
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**

  ```java
  public class ApplicationClass implements IODPConfigurationChangeListener{
      public void onConfigurationChange(int PropertyID, String Property){
          // some application related processing
      }
  }
  ```

**CUSTOMIZATION_RESOURCES variable**

**Syntax**

```java
int CUSTOMIZATION_RESOURCES
```

**PROXY_APPLICATION_ENDPOINT variable**

**Syntax**

```java
int PROXY_APPLICATION_ENDPOINT
```

**PROXY_PUSH_ENDPOINT variable**

**Syntax**

```java
int PROXY_PUSH_ENDPOINT
```

**PWDPOLICY_CHANGED variable**

**Syntax**

```java
int PWDPOLICY_CHANGED
```

**PWDPOLICY_DEFAULT_PASSWORD_ALLOWED variable**

**Syntax**

```java
int PWDPOLICY_DEFAULT_PASSWORD_ALLOWED
```

**PWDPOLICY_ENABLED variable**

**Syntax**

```java
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_FAILUER(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 IODPHTTPAuthChallengeListerner{
    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**

```java
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**

```java
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**

  ```java
  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**

```java
public interface IODPUserRegistrationListener
```

**onAsyncRegistrationResult(boolean, int, String) method**

Callback method that is invoked with the result of user registration.

**Syntax**

```java
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

```java
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**

```java
public class ODPException
```

**ODPException(int) constructor**

Constructs a UserManagerException with specified error code.

**Syntax**

```java
ODPException ( int errorCode )
```

**ODPException(int, String) constructor**

Constructs a UserManagerException with specified error code and error message.

**Syntax**

```java
ODPException ( int errorCode , String message )
```

**ODPException(String) constructor**

Constructs a UserManagerException with specified error message.

**Syntax**

```java
ODPException ( String message )
```

**getErrorCode() method**

Returns the error code of this UserManagerException.

**Syntax**

```java
int getErrorCode ()
```

**ANY_INPUT_FIELD_NULL variable**

User registration fields are null.

**Syntax**

```java
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**

```java
final int REGISTRATION_FAILED_UNKNOWN_ERROR
```

**REGISTRATION_LISTENER_NULL variable**  
Asynchronous user registration listener is unregistered with UserManager.

**Syntax**

```java
final int REGISTRATION_LISTENER_NULL
```

**SINGLETON_INITIALIZATION_FAILED variable**  
Singleton initialization failed.

**Syntax**

```java
final int SINGLETON_INITIALIZATION_FAILED
```

**ODPUserManager class**  
Single entry point for all user on-boarding interactions.

**Syntax**

```java
public class ODPUserManager
```

**Remarks**  
It exposes methods to manage the user credentials.

**cleanUp() method**  
Clear the persistent storage.

**Syntax**

```java
void cleanUp()
```

**Examples**

- **Example 1**

  ```java
  ODPUserManager.cleanup();
  ```

**deleteUser() method**  
Deletes a registered application connection.

**Syntax**

```java
void deleteUser() throws ODPException
```
Exceptions

- ODPException class –

Examples

- Example 1

```java
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 communicate with any host.

**Syntax**

```java
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

```java
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpuserRegistrationManager.enableHTTPS(useHTTPS);
```

**Usage**

For example, Relay server.

**getInstance() method**

Return an object of ODPUserManager.

**Syntax**

```java
ODPUserManager getInstance () throws ODPException
```
**Returns**
ODPUserManager object

**Exceptions**
- ODPException class –

**Examples**
- Example 1

```java
ODPUserManager.initInstance();
ODPUserManager oum=ODPUserManager.getInstance();
```

**initInstance(String) method**
Initialize the ODPUserManager.

**Syntax**
```java
void initInstance (String appID) throws ODPException
```

**Parameters**
- appID – Application identifier.

**Exceptions**
- ODPException class –

**Examples**
- Example 1

```java
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**
```java
boolean.isUserRegistered ()
```

**Returns**
- True - if the user is already registered
- False - if the user is not registered
Exceptions

- ODPException class –

Examples

- Example 1

  ```java
  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

```java
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

  ```java
  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**

```java
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**
  
  ```java
  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**

```java
void setHttpHeaders (Hashtable param1, Hashtable param2) throws ODPException
```

**Parameters**

- **param1** - HTTP header
- **param2** - Cookies

**Exceptions**

- **ODPException class** -

**Examples**

- **Example 1**
  
  ```java
  ODPUserManager.initInstance(appId);
  ODPUserManager odpUserRegistrationManager =
  ```
setRelayServerURLTemplate(String) method

Invoke this method to set the property "ADVANCED_RELAY_SVR_URL_TEMPLATE".

**Syntax**

```java
void setRelayServerURLTemplate(String URLSuffix) throws ODPException
```

**Parameters**

- **URLSuffix** – The value for property ADVANCED_RELAY_SVR_URL_TEMPLATE.

**Exceptions**

- **ODPException class** –
- **MessagingClientException** –

**Examples**

- **Example 1**

```java
ODPUserManager.getInstance();
odpsuserRegistrationManager.setHttpHeaders(param1,param2);

setRelayServerURLTemplate(URLSuffix);
```

setUserRegistrationListener(ODPClientListeners.IODPUserRegistrationListener) method

Set the listener object that is notified with the result of user registration.

**Syntax**

```java
void setUserRegistrationListener(ODPClientListeners.IODPUserRegistrationListener registrationListener)
```

**Parameters**

- **registrationListener** – Object implements the IUserRegistrationListener interface and is invoked with the result of user registration.

**Examples**

- **Example 1**

```java
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
ODPUserManager.getInstance();
odpsuserRegistrationManager.setRelayServerURLTemplate(URLSuffix);
```

```java
ODPUserManager.initInstance(appId);
ODPUserManager odpUserRegistrationManager =
```

CHAPTER 2: Developing BlackBerry Applications

ODPUserManager.getInstance();
odpuserRegistrationManager.setUserRegistrationListener(registrationListener);
CHAPTER 3 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-through 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:

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

.jar files 8

A

Android 1
ANY_INPUT_FIELD_NULL
variableODPException class [ODP SDK API Reference for BlackBerry API] 80
APPLICATION_ID_NULL
variableODPException class [ODP SDK API Reference for BlackBerry API] 81
APPLICATION_USER_ALREADY_REGISTERED
variableODPException class [ODP SDK API Reference for BlackBerry API] 81
APPLICATION_USER_NOT_REGISTERED
variableODPException class [ODP SDK API Reference for BlackBerry API] 81
AppSettingsStore class [ODP SDK API Reference for BlackBerry API] description 61

B

Basic authentication 15
BES provisioning 20
BlackBerry 1, 21, 22, 28, 29, 33, 39–41
provisioning options 19
BlackBerry Developer Environment 6
BlackBerry Java Plug-in for Eclipse 6
BlackBerry JDE, downloading 7
BlackBerry simulator
downloading 7

C

Cache 1, 28
Configuration 39
Connectivity 1, 33
CUSTOMIZATION_RESOURCES
variableIODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] 75

D

data vault
creating 42

documentation roadmap 3

E

EMPTY_RESPONSE_FROM_SERVER
variableODPException class [ODP SDK API Reference for BlackBerry API] 81

I

INTERNAL_PARSING_ERROR
variableODPException class [ODP SDK API Reference for BlackBerry API] 81
INVALID_MODULE_ID variableODPException class [ODP SDK API Reference for BlackBerry API] 81
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] CUSTOMIZATION_RESOURCES
variable 75
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] description 74
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PROXY_APPLICATION_ENDPOINT
variable 75
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PROXY_PUSH_ENDPOINT variable 75
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_CHANGED variable 75
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_DEFAULT_PASSWORD_ALLOWED variable 75
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_ENABLED variable 75
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_EXPIRES_IN_N_DAYS variable 76
Index

IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_HAS_DIGITS variable 76
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_HAS_LOWER variable 76
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_HAS_SPECIAL variable 76
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_HAS_UPPER variable 76
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_LENGTH variable 76
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_LOCK_TIMEOUT variable 76
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_MIN_UNIQUE_CHARS variable 76
IODPConfigurationChangeListener interface [ODP SDK API Reference for BlackBerry API] PWDPOLICY_RETRY_LIMIT variable 77
IODPHTTPAuthChallengeListener interface [ODP SDK API Reference for BlackBerry API] description 77
IODPHttpErrorListener interface [ODP SDK API Reference for BlackBerry API] description 78
IODPPushNotificationListener interface [ODP SDK API Reference for BlackBerry API] description 79
IODPUserRegistrationListener interface [ODP SDK API Reference for BlackBerry API] description 79

iOS 1

J

JSON_PARSING_FAILED variable ODPException class [ODP SDK API Reference for BlackBerry API] 81

L

Logger 40

O

OData for SAP Products 1
OData SDK Components 1, 21, 22, 28, 29, 33, 39–41
ODP SDK API Reference for BlackBerry API 60
ODP SDK API Reference for BlackBerry API AppSettingsStore class 61
ODP SDK API Reference for BlackBerry API client package 60
ODP SDK API Reference for BlackBerry API IODPConfigurationChangeListener interface 74
ODP SDK API Reference for BlackBerry API IODPHttpAuthChallengeListener interface 77
ODP SDK API Reference for BlackBerry API IODPHttpErrorListener interface 78
ODP SDK API Reference for BlackBerry API IODPPushNotificationListener interface 78
ODP SDK API Reference for BlackBerry API IODPUserRegistrationListener interface 79
ODP SDK API Reference for BlackBerry API ODPAppSettings class 62
ODP SDK API Reference for BlackBerry API ODPCertificateManager class 65
ODP SDK API Reference for BlackBerry API ODPCertInfo class 65
ODP SDK API Reference for BlackBerry API ODPClientConnection class 69
ODP SDK API Reference for BlackBerry API ODPClientListeners class 74
ODP SDK API Reference for BlackBerry API ODPException class 80
ODP SDK API Reference for BlackBerry API ODPHTTPAuthChallengeCredentials class 74
ODP SDK API Reference for BlackBerry API ODPUserManager class 82
ODPAppSettings class [ODP SDK API Reference for BlackBerry API] description 62
ODPCertificateManager class [ODP SDK API Reference for BlackBerry API] description 65
ODPCertInfo class [ODP SDK API Reference for BlackBerry API] description 65
ODPClientConnection class [ODP SDK API Reference for BlackBerry API] description 69
ODPClientListeners class [ODP SDK API Reference for BlackBerry API] description 74
ODPException class [ODP SDK API Reference for BlackBerry API] ANY_INPUT_FIELD_NULL variable 80
ODPException class [ODP SDK API Reference for BlackBerry API] APPLICATION_ID_NULL variable 81
ODPException class [ODP SDK API Reference for BlackBerry API] APPLICATION_USER_ALREADY_REGISTERED variable 81
ODPException class [ODP SDK API Reference for BlackBerry API] APPLICATION_USER_NOT_REGISTERED variable 81
ODPException class [ODP SDK API Reference for BlackBerry API] EMPTY_RESPONSE_FROM_SERVER variable 81
ODPException class [ODP SDK API Reference for BlackBerry API] INTERNAL_PARSING_ERROR variable 81
ODPException class [ODP SDK API Reference for BlackBerry API] INVALID_MODULE_ID variable 81
ODPException class [ODP SDK API Reference for BlackBerry API] JSON_PARSING_FAILED variable 81
ODPException class [ODP SDK API Reference for BlackBerry API] REGISTRATION_FAILED_UNKNOWN_ERROR variable 82
ODPException class [ODP SDK API Reference for BlackBerry API] REGISTRATION_LISTENER_NULL variable 82
ODPException class [ODP SDK API Reference for BlackBerry API] SINGLETON_INITIALIZATION_FAILED variable 82
ODPHTTPAuthChallengeCredentials class [ODP SDK API Reference for BlackBerry API] description 74
ODPUUserManager class [ODP SDK API Reference for BlackBerry API] description 82

P
Parser 1, 22
Persistence 1, 29
provisioning options
BlackBerry 19
PROXY_APPLICATION_ENDPOINT variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 75
PROXY_PUSH_ENDPOINT variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 75
PWDPOLICY_CHANGED variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 75
PWDPOLICY_DEFAULT_PASSWORD_ALLOred variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 75
PWDPOLICY_ENABLED variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 75
PWDPOLICY_EXPIRES_IN_N_DAYS variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 76
PWDPOLICY_HAS_DIGITS variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 76
PWDPOLICY_HAS_LOWER variable IODPConfigurationChangeLister interface [ODP SDK API Reference for BlackBerry API] 76

OData App Development using Messaging Channel 95
Index

ner interface [ODP SDK API Reference for BlackBerry API] 76

PWDPOLICY_HAS_SPECIAL
variableIODPConfigurationChangeListe
ner interface [ODP SDK API Reference for BlackBerry API] 76

PWDPOLICY_HAS_UPPER
variableIODPConfigurationChangeListe
ner interface [ODP SDK API Reference for BlackBerry API] 76

PWDPOLICY_LENGTH
variableIODPConfigurationChangeListe
ner interface [ODP SDK API Reference for BlackBerry API] 76

PWDPOLICY_LOCK_TIMEOUT
variableIODPConfigurationChangeListe
ner interface [ODP SDK API Reference for BlackBerry API] 76

PWDPOLICY_MIN_UNIQUE_CHARS
variableIODPConfigurationChangeListe
ner interface [ODP SDK API Reference for BlackBerry API] 76

PWDPOLICY_RETRY_LIMIT
variableIODPConfigurationChangeListe
ner interface [ODP SDK API Reference for BlackBerry API] 77

R

REGISTRATION_FAILED_UNKNOWN_ERROR
R variableODPException class [ODP SDK API Reference for BlackBerry API] 82

REGISTRATION_LISTENER_NULL
variableODPException class [ODP SDK API Reference for BlackBerry API] 82

revert proxy 89

S

SAP Passport 41
SDMCommon 21
signing 19
SINGLETON_INITIALIZATION_FAILED
variableODPException class [ODP SDK API Reference for BlackBerry API] 82

Supportability 1, 40