BlackBerry Workspaces™

Android SDK Developer's Guide

BlackBerry 2021-10-27

Table of Contents

Requirements

Audience

The intended audience for this guide is developers of web-based applications to connect to the Workspaces services.

Required Knowledge

The developers should be familiar with the HTTP protocol and JSON formats for HTTP messages.

Prerequisites

In order to use the Workspaces Android SDK you must have an organization account in the Workspaces cloud service or have an on-premises Workspaces server (deployed as a virtual appliance).

An organization administrator account will be set up by BlackBerry.

Introduction

This guide explains how developers can use the Workspaces Android SDK to develop applications allowing end users to work with files protected by Workspaces.

Workspaces allows users to securely share files with others. File owners maintain full control of each file that they share, including permissions to view, print, copy and download a file. For example, file owners can change access permissions, set a file expiration date, or revoke access to a file at any time even after a file is shared with devices beyond your organization's control.

Workspaces consists of two core services

1. Workspaces: into which files can be uploaded to be securely shared with others
2. Send: by which files are securely shared with others

API Version

This document refers to Workspaces API version 3.0.

Workspaces model

The Workspaces platform is a web-based service that may be hosted on Workspaces cloud-based servers or locally on virtual appliances at an organization (on-premises).

User Types

Workspaces has the following types of users:

Organization Administrator

An administrator that has access to all workspaces in the organization's account, with the ability to create, remove, and modify users (including other administrators), workspaces, groups and all other entities associated with the organization's account. The first Organization Administrator is defined by BlackBerry when the account is first created. The Organization Administrator cannot view documents within the account. There is more than one type of organization administrator. For more information about the different types of organization administrators, see the Workspaces web application.

Workspace Administrator

An administrator for a workspace or group of workspaces within an organization that can view all files in these workspaces, add groups and users, and upload files.

Contributor

A user with the ability to manage content in a workspace (for example, view, upload and delete files).

Visitor

Someone who is not a user of the service that can view files that they receive from a Workspaces user. The file can be viewed in protected format only. A visitor cannot upload files to a workspace or update files in a workspace.

Resources

The Workspaces API is divided into the following categories, each relating to a distinct part of the service:

Files - The following resources are used to manage the Workspaces Send feature for an organization (a service for ad-hoc sending and receiving of document securely).

Resource	Description
Upload & download	Upload files to the Workspaces server where the files can be shared from and download files received from other Workspaces users
Send	Send files by email to recipients
Enumerate	List the files sent via Workspaces
Manage permissions	Set or change access permissions for files sent via Workspaces
Search	Search for files by text or metadata
Track	View audit and tracking information for activities on files sent and received via Workspaces

Organization administration - The following resources are used to manage the workspaces, Workspaces Inbox, and Sent Items in an Organization account.

Resource	Description
Users	Create, update, and remove the organization's users
Roles	Assign roles to users (e.g. workspace Administrator, Contributor)
Aliases	Set email aliases which enable a user to view, in a single session, all files received under different email addresses
Distribution lists	Setup and manage distribution lists of users. Distribution Lists are named sets of users defined globally at the organization level. They can be used as an alias for sending files or be referred to by workspace groups.
Tags	Define metadata tags that can be assigned to files in workspaces
Watermarks	Define the watermark template that can be applied to files downloaded from workspaces
Global Policies	Set global access and usage policies

Workspace management - The following resources can be used to manage a single workspace. For example, a single workspace might be used by a group that is collaborating on a common set of files.

Resource	Description
Groups & Users	Set up groups and users of a workspace
Folders	Set up folders in a workspace
Enumerate	List the files, folders and groups in a workspace
Alerts	Configure alerts for new or changes to files that users should be aware of
Upload, download	Upload or download files to a workspace
Permissions	Set access permissions for files in a workspace

Using the API

The Workspaces Android API provides classes for the previously-noted resources which include Authentication, Documents, Rooms, Users, and Organizations, in addition to others. These resource classes provide static methods that allow access to their functionality.

In general, most of the methods require that an authentication token(secure session id, or SSID) be passed in, along with the appropriate parameter(s)- often a JSON object that represents the data the resource method needs. Successful requests will generally return a JSON object or InputStream(e.g. when downloading documents) to indicate that the requested action has been successfully performed. A common JSON object returned is the BulkOperationResultJson that provides details on the success of each operation or any problem(s) encountered.

Here's a simple example that illustrates those ideas. It uses an instance of ApiSession class to start a service account session for a given user. StartSessionWithServiceAccount method of Apisession requires the user's email address for which the service account has been created, the issuer(an ID created when the service account is created), the token expiration in minutes, the PrivateKey from the keystore, and a string representing the encryption algorithm(e.g "SHA1withRSA").It will return a "Success" LoginResult if the session starts successfully.The ApiSession then sends a request , to one of the getters to the resources, to get an instance of a resource. In the following code sample, the listRoomsV30 of workspaces object gets back an iterable JSON object representing a list of WorkspaceInfoJson(with each item inside representing a workspace).

```
APISession apiSession = new APISession(serverUrl, null);

Workspaces workspaces = apiSession.getWorkspacesResource();

LoginResult loginResult = apiSession.startSessionWithServiceAccount(username,
                                                                    serviceAccount,
                                                                    expiresInMinutes,
                                                                    pKey,
                                                                    "SHA256withRSA");

ItemListJson<WorkspaceInfoJson> itemListJson = workspaces.listRoomsV30(addExternalData,
                                                                       adminMode,
                                                                       includeSyncData,
                                                                       includeWSPolicyData,
                                                                       workspaceTypes);
```

Many domain-specific errors will result in a WatchdoxSDKException being returned to the caller. It has a getErrorText() method to get information about the error. More-generalized Java errors like IllegalStateException are also possible.

Examples for using the Rooms resource and other resources can be found in the Examples section.

Authentication using Service Accounts

Service accounts provide an alternative means of authenticating requests sent to the Workspaces server. Using a service account removes the need to have a user's password in order to authenticate a user. When using a Service Account you can configure Workspaces to allow groups of users based on their email domain(s) and/or specific users access to their Workspaces accounts.

6.1 Steps for creating a Service Account

1. Create or obtain a SSL certificate. There are a number of ways to do this including using tools like OpenSSL or Java's Keytool application to create a Self-Signed certificate, or you can purchase a commercial certificate from any number of certificate providers.
2. Extract the Public key from the certificate. You'll need this to paste the Public key into the Workspaces Admin console.
3. Configure the Service Account in Workspaces.

   While logged in to the Workspaces Admin Console as an administrator:

   a. Navigate to Service Accounts under Authentication on the left-hand side, and click the + icon.

   b. Under System accounts:

      i. In the **Public key** field, copy and paste the contents of the Public key. Depending on how the certificate was generated and the public key displayed, the key may be bracketed by a set of tags such as -----*BEGIN PUBLIC KEY*----- and -----*END PUBLIC KEY*-----. Do not include these tags when copying the pubilc key.

      ii. In the **System acounts** field, enter a list of user email addresses that will be allowed to authenticate using this service account. Separate each address with a space. If you only want to authenticate groups of users using their email domain this field may be left blank.

      iii. In the **Domain system accounts** field, enter a list of email domains that will be allowed to authenticate using this service account. Separate each domain with a space. If you only want specific users to be able to authenticate this field may be left blank.

      iv. In the **Algorthm** dropdown, select the algorithm that was used to create the certificate.

      v. Click Apply to save the Service Account configuraiton.

4. In your application code:

   a. Create a valid, signed Workspaces Authentication token by using the private key from the certificate.

   b. Use the generated token in an Authorization header in your request to the Workspaces server.

6.2 Authenticating using startSessionWithServiceAccount

Once the Service Account has been created in the Workspaces Admin Console clients can connect using the certificate. The startSessionWithServiceAccount method on APISession provides the means to start a session using the Service Account. This method will construct an authentication token and validate that the specified user has access to use the service account to make requests to the server. If they do then LoginResult will return with "Success". For further details see Example 1.

```
APISession apiSession = new APISession(serverUrl, null);

LoginResult loginResult = apiSession.startSessionWithServiceAccount(username,
                                                                    serviceAccount,
                                                                    expiresInMinutes,
                                                                    pKey,
                                                                    "SHA256withRSA");
```

6.3 Using OpenSSL to create a certificate

Run the openssl applicaiton to create a certificate and private key file. In the example below **<PRIVATEKEY>** represents the name of the file where the private key will be stored. **<CERTIFICATE>** represents the name of the file where the certificate will be stored.

When you run the openssl command you will be prompted for several pieces of information used in creating the certificate.

```
openssl req -newkey rsa:2048 -nodes -keyout <PRIVATEKEY> -x509 -out <CERTIFICATE>
```

Run openssl again to display the public key.

```
openssl rsa -in <PRIVATEKEY> -pubout
```

The output will look something like the example shown below. The public key is the text shown between the -----*BEGIN PUBLIC KEY*----- and -----*END PUBLIC KEY*----- tags.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0X43UwF1exJMv8JktJGa
XIYOwARj/w95tvYuGiY42pTwH8Ttp8eYlwAX3bT5awdC/D7qLz2oEWIMb8QH+0qF
L7KU0nBHzyWBIqgjJKywegbsFuKXHxlMZrGkcaAmIiQ0VxesZyxtWzPlHvvX2i67
kAygyZ2VCgj/D7KZXluLV55XY/vH44ohgPu18D3mbwX8pTWqfaOeUQUzv4kIWwta
yDiQu4+ec+sr47zNNzBUCYoAR99+2b/anmxdrOn8/QJcCu6zWBzlQGyXK5fhI5tA
18AC32rKBkV/hLhIM5D7n3JjQ73hwiUcqct85gl4Nf9YowUGC3h1ejuhKf4VYah/
KQIDAQAB
-----END PUBLIC KEY-----
```

6.4 Using Keytool to create a certificate

Run the Java keytool application to create a certificate and insert it into a Java KeyStore. **<ALIAS>** in the example below represents the name or alias that will later be used to retrieve the certifcate from the keystore. **<KEYSTORE FILE NAME>** represents the name of the KeyStore file where the certificate will be stored.

You will be prompted for several pieces of information that will be used to create the certificate. You will also be prompted for a password to secure the keystore. Make a note of this password as you will need it in the next step.

```
keytool -genkey -alias <ALIAS> -keyalg RSA -keystore <KEYSTORE FILE NAME> -keysize 2048
```

Run the keytool application again piping the result through OpenSSL to display the public key information. In the example below the command would be on a single line. It is wrapped across lines here for readability.

```
keytool -list -rfc -keystore <KEYSTORE FILE NAME> -alias <ALIAS> \
-storepass <KEYSTORE PASSWORD> | openssl x509 -inform pem -pubkey -noout
```

The resulting output will look something like the example shown below. The public key is the text shown between the -----*BEGIN PUBLIC KEY*----- and -----*END PUBLIC KEY*----- tags.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAy+ZwUOXSlAnq5oM6qGZC
yCt0PfJVY9lyEr4jaRGXJMPTohegS2qonDkH9+yCYDtJ/8hz3LZ4PPedhG99pddG
kBp5uUdPETJTQv10vOtov4shMbs8wQwATumnn9Hcgu3edITJWSKirSYAoslTRg4P
/Q1nHOaGv+vARRZ4wwuJylthSMlH+0RALkYLHu11X38iLNK1zdM3nOxgN7ldqOsH
Y/Ub/lTFD9Q8sCWMjPHGGk0hYjla1CbkJOoUi5xPrFvph6J15nTYOdZcua41R0Et
A9PGEE9rGNuIAtGV1kknoXXjejf356R7LP/iIsWE66ozoJiKkhWJA3lrrGD0kpNa
vwIDAQAB
-----END PUBLIC KEY-----
```

See "Example 1: Connect and authenticate a user" for an example of obtaining an authentication token via a service account once the above steps are completed.

Authentication using OAuth

Workspaces OAuth lets users allow other applications to interact with their Workspaces account without providing that other application their Workspaces authentication credentials, and for only a time-limited basis. For example, a sales management application can integrate Workspaces functionality to allow the controlled sharing of documents via Workspaces, within the sales management app, without requiring users to provide their Workspaces usernames and passwords to the sales management app.

7.1 Registering a client with a Workspaces server

Someone wishing to use Workspaces OAuth must register with Workspaces and obtain a client ID and client secret that will be passed to Workspaces in order to obtain a Workspaces authentication token.

7.2 Authenticating using startSessionWithOAuth

The startSessionWithOAuth method in APISession provides an easy way to authenticate users using OAuth. After creating an instance of APISession in the Activity and making a call to the startSessionWithOAuth method, a window will appear to accept the authentication credentials for the user. This window will remain visible until the user successfully authenticates with the Workspaces server or the user closes the window. Because of this the startSessionWithOAuth method is a blocking call. Once the user has provides their credentials correctly or closes the window.

```
APISession apiSession = new APISession(serverUrl, null);

apiSession.startSessionWithOAuth(this, email, refreshToken, showUiIfRefreshFails);
```

The "LOGINRESULT" stringExtra and "APISESSION" serializableExtra can be retrieved from onActivityResult method Intent. If the user provided a correct set of credentials the "LOGINRESULT" will contain a "Success" LoginResult. After the successful LoginResult , we can retrieve the "APISESSION" to be used for accessing the resources.

```
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        super.onActivityResult(requestCode, resultCode, data);

        if(data.hasExtra("LOGINRESULT")) {
            if (data.getStringExtra("LOGINRESULT").contentEquals("Success")) {
                apiSession = (APISession) data.getSerializableExtra("APISESSION");
            }
        }
```

7.3 Manually authenticating using OAuth

The basic flow is(WDX_URL means the base URL of the Workspaces server being used):

1. Make an unauthenticated call to *WDX_URL*/api/3.0/authentication/parameters to get the authentication URI's. In the returned values, authorizationUri is the URI for the temporary token in #2 below; accessTokenUri is the URI for obtaining the full token in #4 below.
2. Direct the user to the Workspaces authenticationUri (e.g. via a redirect in your own web app), passing the proper params(see below). One of these params is the redirect URI in your app to which the user should be sent after authenticating with Workspaces. It will look something like this (wrapped across lines here, but one single line when used):

```
<WDX_URL>/<AUTHORIZATION_URI>?response_type=code&client_id=<CLIENT_ID>&locale=en_US
&redirect_uri=<REDIRECT_URI>
```

3. When that redirect URI is serviced in your web app, a temporary code is included on the URL that will be used to obtain a valid auth token. It will look something like this:

```
<REDIRECT_URI>/?code=219e5a32-d74f-473a-91a4-fd74f95e091c&locale=en-us
```

4. Make a call to the Workspaces accessTokenUri to obtain a valid auth token, refresh token, and an expiration value for the auth token. That auth token can be used to authenticate subsequent API calls to the Workspaces server (wrapped across lines here, but one single line when used).

```
<WDX_URL>/<ACCESS_TOKEN_URI>?client_id=<CLIENT_ID>&redirect_uri=<REDIRECT_URI>&
client_secret=<CLIENT_SECRET>&grant_type=authorization_code&code=<AUTH_CODE>
```

5. That call will return JSON that contains an access token, expiration, and refresh token. The access token can be used in the authorization header of subsequent calls.

Authentication using an existing token

In some situations a developer may want to create a separation of concerns whereby one module might be responsible for obtaining authentication tokens from the Workspaces server while other components use those tokens to communicate with the Workspaces server. The Workspaces SDK provides a means to do this using the *loadExistingSession* method on APISession.

In the module that provides the authentication tokens there would be some code, similar to the snippet below, which obtains a valid authentication token for the Workspaces server.

```
pubic String ObtainTokenFromSeparateModule(String username, KeyStore keystore)
{
  String authToken = String.Empty;

  String serviceAccountName = "com.watchdox.system.xxxx.yyyy";
  int expiresInMinutes = 5;

  // Get certificate of public key
  Certificate cert = (Certificate) keystore.getCertificate(alias);
  // Get public key
  PublicKey publicKey = cert.getPublicKey();
  pKey = new KeyPair(publicKey, (PrivateKey) key).getPrivate();

  APISession apiSession = new APISession(serverUrl, null);;

  LoginResult loginResult = apiSession.startSessionWithServiceAccount(username,
                                                                      serviceAccount,
                                                                      expiresInMinutes,
                                                                      pKey,
                                                                      "SHA256withRSA");

  if (loginResult == LoginResult.SUCCESS) {
    authToken = apiSession.GetToken();
  }

  return authToken;
}
```

Having obtained an authentiation token from the other module, it would then passed to the loadExistingSession method of an APISession instance.

```
String authToken = ObtainTokenFromSeparateModule();

if (!string.IsNullOrEmpty(authToken))
{
  APISession apiSession = new APISession(serverUrl, null);
  LoginResult loginResult = apiSession.loadExistingSession(authToken);
}
```

Using System Properties

9.1 SDK properties

The BlackBerry Workspaces Android SDK uses several properties for system-level configuration. By default the SDK will look for a properties file named **workspaces-sdk.properties** along the classpath.The properties recognized by SDK are as follows.

sdk.apiAddress

The root URL for the Workspaces API server. The API address should form a fully qualified URL with the protocol, host name and the "/api" path. For example: "https://myworkspacedomain.server.com/api". **Note:** *The API server address can also be set using the APIRunner.setApiHost() or APIRunner.setApiAddress() methods.*

```
APIRunner.setApiHost ("myworkspacedomain.server.com");
  or
APIRunner.setApiAddress ("https://myworkspacedomain.server.com/api");
```

Default value: null

sdk.log.enable

Controls whether logging information is generated by the SDK. When enabled the SDK will log information about requests it sends out to the API server and the responses it receives back. **Note:** *Logging can also be enabled or disabled using the APIRunner.enableLogging() method.*

```
APIRunner.enableLogging (true, null);
  or
APIRunner.enableLogging (true, "/log/workspacesSDKOutput.txt");
```

Default value: false

sdk.log.outputLocation

Sets the location where the SDK will write log information when logging is enabled. **Note:** *The log output location can also be set using the APIRunner.enableLogging() method.*

```
APIRunner.enableLogging (true, "/log/workspacesSDKOutput.txt");
```

Default value: /logs/workspacesSDKOutput.txt

sdk.log4j.external.configuration

Controls how the logging configuration is set. When this property is specified as *true* the standard log4j.properties file will be used to initialize the logging configuration. When the property is not set, or set to *false* the SDK uses its own internal logging configuration.

Default value: false

9.2 Application properties

The SDK will read all of the properties found in the properties file and make them available as system properties. This makes it possible to include properties for an application making use of Workspaces SDK within SDK's properties file. To access any of the properties set in the SDK properties file call the System.getProperty() method.

```
System.getProperty("one.of.my.app.properties");
```

9.3 Using a different properties file

It's also possible to use a different properties file than the default **workspaces-sdk.properties**. To do this you must specify the desired properties file on the Java command line via the **workspaces.sdk.properties** property.

In specifying the properties file, the filename should be prefixed by an identifier that indicates where the SDK should look for the file. A prefix of *classpath:* indicates that the SDK should look in the classpath for the file. A prefix of *file:* indicates that the SDK should look in the file system for the file. When using *file:*, a fully qualified filename or an relative path can be used.

```
java classname -Dworkspaces.sdk.properties=file:c:\myApp\config\myProperties.properties
  or
java classname -Dworkspaces.sdk.properties=classpath:myProperties.properties
```

Examples

This section describes examples of common application scenarios, to illustrate how to use the API.

Note also that the term room is interchangeable with Workspace.

Example 1: Connect and authenticate a user

This example provides how to connect and authenticate to the server. Basically there are two methods supported:

i. **Authenticating using a Service Account:** uses a certificate to sign the authentication credentials and is best used with server to server applications.

ii. **Authenticating using OAuth:** is best used for client to server applications that allows a user to authenticate without exposing their credentials to the client application.

In the following sections, its depicted in details how to authenticate a user to the Workspaces' services.

11.1 Authenticating using a Service Account

To Start a session using service account, you must first know the username or email address of the user the request will be made on the behalf of, the issuer id for the Service account, and determine for how long the signed token will be valid. You will also need the PrivateKey from the keystore, and a string representing the encryption algorithm(e.g "SHA1withRSA"). It assumes that you have already copied the public key to the Workspaces server(see the section on "Using Service Accounts" for more information on that). How an application determines or aquires a user's email address would be up to the application. The issuer is the identifier assigned when the service account was created in the Workspaces Admin console. This issuer would be in the format *com.watchdox.system.xxxx.yyyy* where xxxx and yyyy are some set of numbers and letters- for example, *com.watchdox.system.fadd.3015*. The expiration time represents the number of minutes for which the token should be valid.

With this information in hand and creating an instance of APISession, a simple call to a startSessionWithServiceAccount method in ApiSession will return the loginResult.The server returns a "Success" LoginResult if the session starts successfully:

```
APISession apiSession = new APISession(serverUrl, null);

LoginResult loginResult = apiSession.startSessionWithServiceAccount(username,
                                                                    serviceAccount,
                                                                    expiresInMinutes,
                                                                    pKey,
                                                                    "SHA256withRSA");
```

11.2 Authenticating using OAuth

To Start a session using OAuth, you must first know the email address to pass to service provider, null or empty if not known and to be determined during the OAuth process. Its also optional to provide refresh token and showUiIfRefreshFails(indicates whether to show UI if refresh token exists but the refresh fails).

After creating an instance of APISession and making startSessionWithOAuth method call, an OauthBrowser window pop ups to receive an authentication email and password. Once the user provides a correct email and password, the pop up widow will close and the startSessionWithOAuth method returns a "Success" LoginResult if the session starts successfully.

```
APISession apiSession = new APISession(serverUrl, null);

LoginResult loginResult = apiSession.startSessionWithOAuth(this,
                                                           email,
                                                           refreshToken,
                                                           showUiIfRefreshFails);
```

Example 2: Add users to a workspace

This example adds a user to a Workspace. It requires three separate method calls. (Note the different permissions required for each step).

12.1 Create Workspace (can only be done by an organization admin).

This returns a RoomsJson object that represents the room that was created.

```
Workspaces workspaces = apiSession.getWorkspacesResource();

// Create the JSON object needed for the method call, and set its values.
CreateRoomJson createRoomJson = new CreateRoomJson();
createRoomJson.setName(name);
createRoomJson.setDescription(description);
createRoomJson.setAdministrators(administrators);

// Using the auth token from Example 1, call the method, and get a JSON object back
RoomJson roomJson = workspaces.createRoomV30(createRoomJson);
```

12.2 Create new room group (can only be done by a Workspace admin)

```
 Workspaces workspaces = apiSession.getWorkspacesResource();

// Create a JSON object the represents the new group and set its values
PermittedEntityFromUserJson permittedEntityFromUserJson =
    new PermittedEntityFromUserJson();
permittedEntityFromUserJson.setAddress(groupName);
permittedEntityFromUserJson.setEntityType(EntityType.GROUP);

// The JSON object needed for the resource method call
AddEntityVdrJson addEntityVdrJson = new AddEntityVdrJson();

// Create a new permissions JSON with default values
PermissionFromUserJson permissionFromUserJson = new PermissionFromUserJson();

addEntityVdrJson.setPermittedEntity(permittedEntityFromUserJson);
addEntityVdrJson.setNewPermissions(permissionFromUserJson);

// Make the call to the Rooms resource and return "success"
String result = workspaces.addEntityV30(workspaceId, addEntityVdrJson);
```

12.3 Add members (users) to group

```
Workspaces workspaces = apiSession.getWorkspacesResource();

List<AddMemberToGroupJson> memberList = new ArrayList<AddMemberToGroupJson>();

// Loop through the List<String> userAddresses & make a AddMemberToGroupJson for each user
for (String currentAddress : userAddresses)
{
  PermittedEntityFromUserJson currentEntity = new PermittedEntityFromUserJson();
  currentEntity.setAddress(currentAddress);
  currentEntity.setEntityType(EntityType.USER);

  AddMemberToGroupJson currentMemberJson = new AddMemberToGroupJson();
  currentMemberJson.setEntity(currentEntity);

  memberList.add(currentMemberJson);
}

AddMembersToGroupWithGroupJson groupMemberJson = new AddMembersToGroupWithGroupJson();
groupMemberJson.setMembersList(memberList);

// roomId is the integer identifying the room
groupMemberJson.setRoomId(roomId);

// Set the group name to be a string
groupMemberJson.setGroupName(groupName);

// Make the call to the Rooms resource and return "success"
String result = workspaces.addMembersToGroupV30(groupMemberJson);
```

Example 3: Add permissions for a group of users to access a file

This example allows access to a document for a group of users. It assumes a group and a workspace exist, the latter with a single document, and group ("mygroup") exists; these are described in the preceding example.

13.1 Get list of documents in a Workspace

```
Workspaces workspaces = apiSession.getWorkspacesResource();

// Create a json object for the document search, and accept its defaults.
// Adjust if needed.
ListDocumentsVdrJson listDocumentsJson = new ListDocumentsVdrJson();

// Make the call to get the list of documents
PagingItemListJson<BaseJson> documentList =
    workspaces.listDocumentsV30(roomId, listDocumentsJson);
```

13.2 Create the json object for the permissions

Default to true value for all permission. Set the group name and EntityType.GROUP as well. Note the nested json objects that are created and then set on other, enclosing json objects.

```
PermittedEntityFromUserJson groupPermission = new PermittedEntityFromUserJson();
groupPermission.setAddress(groupName);
groupPermission.setEntityType(EntityType.GROUP);

permissionsList.add(groupPermission);

// Adjust permissions as needed
PermissionSetJson permissionSet = new PermissionSetJson();
permissionSet.setDownloadOriginal(YesNoDefault.YES);
permissionSet.setDownloadControlled(YesNoDefault.YES);
permissionSet.setCopy(YesNoDefault.YES);
permissionSet.setEdit(YesNoDefault.YES);
permissionSet.setPrint(YesNoDefault.YES);
permissionSet.setProgrammaticAccess(YesNoDefault.YES);
permissionSet.setSpotlight(YesNoDefault.YES);
permissionSet.setWatermark(YesNoDefault.YES);

VdrAddPermissionsJson permissionsJson = new VdrAddPermissionsJson();
permissionsJson.setPermittedEntities(permissionsList);
permissionsJson.setPermissionSet(permissionSet);

Set<String> documentGuids = new HashSet<String>();

documentGuids.add(myDoc.getGuid());
permissionsJson.setDocumentGuids(documentGuids);
```

13.3 Add permissions to the group

Set the document GUID from the returned document list and make the call to set the permissions. A BulkOperationResultJson is returned, which has information on the success or failure(with errors encountered) for each document in the list(which in this case was just 1).

```
BulkOperationResultJson result =
    workspaces.addPermissionsV30(roomId, permissionsJson);
```

Example 4: Upload a file

14.1 Upload a file to workspace

This example shows how to upload a document to a workspace. The **UploadManager** uses *UploadDocumentToRoom* method to upload a file to a specific workspace. This method returns uploadResult which contains a detailed information about the uploaded file.

```
// Get an instance of UploadManager
UploadManager uploadManager = apiSession.getUploadManager();

// Create a new SubmitDocumentsVdrJson JSON
SubmitDocumentsVdrJson uploadInfo = new SubmitDocumentsVdrJson();
uploadInfo.setOpenForAllRoom(false);
RoomRecipientsJson recipientsJson = new RoomRecipientsJson();
recipientsJson.setGroups(groups);
recipientsJson.setDomains(domains);
uploadInfo.setRecipients(recipientsJson);
uploadInfo.setFolder(folder);
uploadInfo.setTagValueList(null);
uploadInfo.setDeviceType(DeviceType.SYNC);

// A call to the UploadDocumentToRoom
UploadResult uploadResult = uploadManager.uploadDocumentToRoom(uploadInfo,
    roomId, destinationFileName, filename, null, false);
```

14.2 Send a file via Workspaces

This example is parallel to the preceding one, but applies to the Workspaces Exchange, the **UploadManager** uses *UploadDocument* method to upload a document to the exchange and send a link to it to email recipients. This method also returns uploadResult which contains a detailed information about the uploaded file.

```
// Get an instance of UploadManager
UploadManager uploadManager = apiSession.getUploadManager();

// Create a new SubmitDocumentSdsJson JSON
SubmitDocumentSdsJson uploadInfo = new SubmitDocumentSdsJson();
HashSet<String> documentGuids = new HashSet<String>();
documentGuids.add(uploadManager.getNewGuidForDocument());
    // Create a new permission JSON
PermissionFromUserJson permissionJson = new PermissionFromUserJson();
permissionJson.setCopy(true);
permissionJson.setDownload(true);
permissionJson.setDownloadOriginal(false);
permissionJson.setExpirationDate(new Date());
uploadInfo.setPermission(permissionJson);
uploadInfo.setUserRecipients(userRecipients);
uploadInfo.setActiveDirectoryGroupsRecipients(ADGroupsRecipients);
uploadInfo.setWhoCanView( WhoCanView.RECEIPIENTS_ONLY);
uploadInfo.setDocumentGuids(documentGuids);

// A call to the UploadDocument
UploadResult uploadResult =
    uploadManager.uploadDocument(uploadInfo, localPath, null, filename, null);
```

Example 5: Download a file

This example shows how to download a document using **DownloadManager** . In the following snippets ,its depicted some of the methods used to download a file.

15.1 Using DownloadFileById

In the following snippet the file could be downloaded using *DownloadFileById* method for a specific document Id :

```
// Get an instance of DownloadManager
DownloadManager downloadManager = apiSession.getDownloadManager();

// A call to the DownloadFileById
try {
    downloadManager.downloadFileById(docId, "", roomId, destinationPath,
            destinationDocName,lastUpdateTime,
            true, true, DownloadTypes.MAX_ALLOWED);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

15.2 Using DownloadFileByName

This method uses Document name to download the file :

```
// Get an instance of DownloadManager
DownloadManager downloadManager = apiSession.getDownloadManager();

// A call to the DownloadFileByName
try {
    downloadManager.downloadFileByName(roomId, folderPath, docName,
        destinationPath, destinationDocName, lastUpdateTime);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

15.3 Using DownloadFileToBuffer

It is also possible to download a file to a buffer using *DownloadFileToBuffer*. This method returns a byte array of the file downloaded.

```
// Get an instance of DownloadManager
DownloadManager downloadManager = apiSession.getDownloadManager();

// A call to the DownloadFileToBuffer
byte[] buffer = downloadManager.downloadFileToBuffer(docId,
        DownloadTypes.ORIGINAL);
```

Example 6: Enumerate files in a workspace

This example will request a list of all documents in a Workspace that the user has access rights to. The server will return a list of documents or folders, which can then be iterated over.

Example 7: Change file permissions

This case shows how permissions can be changed for a document in a Workspace. The document is selected by the group that has access rights to it. In this example, the permissions to be set are: edit, print, spotlight, watermark - No. All other permissions will remain unchanged.

This case assumes the guid for the document is known; if not, a list of documents can be retrieved as shown in Example 3, the desired document selected by name, and the GUID retrieved from it.

```
// Create the json that indicates the group that has permissions to the document
PermittedEntityFromUserJson groupPermission = new PermittedEntityFromUserJson();
groupPermission.setAddress(groupName);
groupPermission.setEntityType(EntityType.GROUP);

List<PermittedEntityFromUserJson> permissionsList =
    new ArrayList<PermittedEntityFromUserJson>();
permissionsList.add(groupPermission);

// Add the document GUID
Set<String> documentGuids = new HashSet<String>();
documentGuids.add(myDoc.getGuid());

// Create the json for the permissions and set them to NO
PermissionSetJson permissionSet = new PermissionSetJson();
permissionSet.setEdit(YesNoDefault.NO);
permissionSet.setPrint(YesNoDefault.NO);
permissionSet.setSpotlight(YesNoDefault.NO);
permissionSet.setWatermark(YesNoDefault.NO);

VdrEditPermissionsJson editPermissionsJson = new VdrEditPermissionsJson();
editPermissionsJson.setPermittedEntities(permissionsList);
editPermissionsJson.setDocumentGuids(documentGuids);
editPermissionsJson.setPermissionSet(permissionSet);

// Make the call and get a BulkOperationResultJson back
BulkOperationResultJson result =
    workspaces.editPermissionsV30(roomId, editPermissionsJson);
```

Example 8: Enumerate folders and workspaces

This example gets a list of all folders and workspaces that the user has access to. There are two steps: get a list of all workspaces and then get a list of all folders in these workspaces.

18.1 Get list of workspaces (rooms) for the user.

```
Workspaces workspaces = apiSession.getWorkspacesResource();

// This returns a list of rooms, which can be iterated over. The other parameters include:
// addExternalData, adminMode, includeSyncData, includeWorkspacePolicyData, and
// workspaceTypes. Please see the javadoc documentation for details.
ItemListJson<WorkspaceInfoJson> itemListJson =
    workspaces.listRoomsV30(true, true, false, false, null);
```

18.2 Get list of folders in a specific Workspace

```
Workspaces workspaces = apiSession.getWorkspacesResource();

// This returns a folder object, which contains details about the current workspace, as
// well as a sub folder list that can be iterated over.
FolderJson folderJson = workspaces.getFolderTreeV30(roomId, null);

List<FolderJson> subFolders = folderJson.getSubFolders();
```

Example 9: Retrieve a list of activities for a named file

This example shows how an activity log is retrieved for a document using the guid. The server will return a list of activity log entries, which can then be iterated over.

Example 10: Add a user to a room group

In this example, a new user list(of one or more users) is added to an existing group. The users' email addresses, room id, and the group name are passed in as parameters.

Request

```
Workspaces workspaces = apiSession.getWorkspacesResource();

List<AddMemberToGroupJson> memberList = new ArrayList<AddMemberToGroupJson>();

// Loop through the List<String> userAddresses & make a AddMemberToGroupJson for each user
for (String currentAddress : userAddresses)
{
  PermittedEntityFromUserJson currentEntity = new PermittedEntityFromUserJson();
  currentEntity.setAddress(currentAddress);
  currentEntity.setEntityType(EntityType.USER);

  AddMemberToGroupJson currentMemberJson = new AddMemberToGroupJson();
  currentMemberJson.setEntity(currentEntity);

  memberList.add(currentMemberJson);
}

AddMembersToGroupWithGroupJson groupMemberJson = new AddMembersToGroupWithGroupJson();
groupMemberJson.setMembersList(memberList);

// roomId is the integer identifying the room
groupMemberJson.setRoomId(roomId);

// Set the group name to be a string
groupMemberJson.setGroupName(groupName);

// Make the call to the Rooms resource and return "success"
String result = workspaces.addMembersToGroupV30(groupMemberJson);
```

Example 11: Delete a file from a workspace

This example deletes one or more documents from a room. The server will return an object that indicates success, or details about any error that may have occurred.

Example 12: Update or remove a file from the Workspaces Inbox or Sent items

This example is similar to the preceding case, but for a document in the Workspaces Exchange.

Legal notice

This documentation including all documentation incorporated by reference herein such as documentation provided or made available on the BlackBerry website provided or made accessible "AS IS" and "AS AVAILABLE" and without condition, endorsement, guarantee, representation, or warranty of any kind by BlackBerry Limited and its affiliated companies ("BlackBerry") and BlackBerry assumes no responsibility for any typographical, technical, or other inaccuracies, errors, or omissions in this documentation. In order to protect BlackBerry proprietary and confidential information and/or trade secrets, this documentation may describe some aspects of BlackBerry technology in generalized terms. BlackBerry reserves the right to periodically change information that is contained in this documentation; however, BlackBerry makes no commitment to provide any such changes, updates, enhancements, or other additions to this documentation to you in a timely manner or at all.

This documentation might contain references to third-party sources of information, hardware or software, products or services including components and content such as content protected by copyright and/or third-party websites (collectively the "Third Party Products and Services"). BlackBerry does not control, and is not responsible for, any Third Party Products and Services including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third Party Products and Services. The inclusion of a reference to Third Party Products and Services in this documentation does not imply endorsement by BlackBerry of the Third Party Products and Services or the third party in any way.

EXCEPT TO THE EXTENT SPECIFICALLY PROHIBITED BY APPLICABLE LAW IN YOUR JURISDICTION, ALL CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS, OR WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY CONDITIONS, ENDORSEMENTS, GUARANTEES, REPRESENTATIONS OR WARRANTIES OF DURABILITY, FITNESS FOR A PARTICULAR PURPOSE OR USE, MERCHANTABILITY, MERCHANTABLE QUALITY, NON-INFRINGEMENT, SATISFACTORY QUALITY, OR TITLE, OR ARISING FROM A STATUTE OR CUSTOM OR A COURSE OF DEALING OR USAGE OF TRADE, OR RELATED TO THE DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN, ARE HEREBY EXCLUDED. YOU MAY ALSO HAVE OTHER RIGHTS THAT VARY BY STATE OR PROVINCE. SOME JURISDICTIONS MAY NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES AND CONDITIONS. TO THE EXTENT PERMITTED BY LAW, ANY IMPLIED WARRANTIES OR CONDITIONS RELATING TO THE DOCUMENTATION TO THE EXTENT THEY CANNOT BE EXCLUDED AS SET OUT ABOVE, BUT CAN BE LIMITED, ARE HEREBY LIMITED TO NINETY (90) DAYS FROM THE DATE YOU FIRST ACQUIRED THE DOCUMENTATION OR THE ITEM THAT IS THE SUBJECT OF THE CLAIM.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, IN NO EVENT SHALL BLACKBERRY BE LIABLE FOR ANY TYPE OF DAMAGES RELATED TO THIS DOCUMENTATION OR ITS USE, OR PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE, HARDWARE, SERVICE, OR ANY THIRD PARTY PRODUCTS AND SERVICES REFERENCED HEREIN INCLUDING WITHOUT LIMITATION ANY OF THE FOLLOWING DAMAGES: DIRECT, CONSEQUENTIAL, EXEMPLARY, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR AGGRAVATED DAMAGES, DAMAGES FOR LOSS OF PROFITS OR REVENUES, FAILURE TO REALIZE ANY EXPECTED SAVINGS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF BUSINESS OPPORTUNITY, OR CORRUPTION OR LOSS OF DATA, FAILURES TO TRANSMIT OR RECEIVE ANY DATA, PROBLEMS ASSOCIATED WITH ANY APPLICATIONS USED IN CONJUNCTION WITH BLACKBERRY PRODUCTS OR SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF BLACKBERRY PRODUCTS OR SERVICES OR ANY PORTION THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR SERVICES, COST OF CAPITAL, OR OTHER

SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES WERE FORESEEN OR UNFORESEEN, AND EVEN IF BLACKBERRY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, BLACKBERRY SHALL HAVE NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO BLACKBERRY AND ITS AFFILIATED COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE PROVIDERS), AUTHORIZED BLACKBERRY DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND THEIR RESPECTIVE DIRECTORS, EMPLOYEES, AND INDEPENDENT CONTRACTORS.
IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF BLACKBERRY OR ANY AFFILIATES OF BLACKBERRY HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with BlackBerry's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for acquiring them. You should not install or use Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and Services that are provided with BlackBerry's products and services are provided as a convenience to you and are provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations, or warranties of any kind by BlackBerry and BlackBerry assumes no liability whatsoever, in relation thereto. Your use of Third Party Products and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other agreements applicable thereto with third parties, except to the extent expressly covered by a license or other agreement with BlackBerry.

The terms of use of any BlackBerry product or service are set out in a separate license or other agreement with BlackBerry applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY BLACKBERRY FOR PORTIONS OF ANY BLACKBERRY PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.

BlackBerry Enterprise Software incorporates certain third-party software. The license and copyright information associated with this software is available at http://worldwide.blackberry.com/legal/thirdpartysoftware.jsp.

BlackBerry Limited
2200 University Avenue East
Waterloo, Ontario
Canada N2K 0A7

BlackBerry UK Limited
200 Bath Road
Slough, Berkshire SL1 3XE
United Kingdom

Published in Canada