BlackBerry Workspaces™

.NET SDK Developer's Guide

BlackBerry 2021-03-29

Table of Contents

Requirements

Audience

The intended audience for this guide is developers of .NET applications wanting to connect to the Workspaces services.

Required Knowledge

Developers should be familiar with creating applications for the .NET platform. Knowledge of the HTTP protocol and JSON formats for HTTP messages will also be useful.

Prerequisites

In order to use the Workspaces .NET SDK you must have an organization account in the Workspaces cloud service or have an on-premises Workspaces installation (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 .NET SDK to develop applications to allow 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

a. Workspaces: into which files can be uploaded to be securely shared with others

b. 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 .NET SDK provides classes for the previously-noted resources which include Authentication, Files, Workspaces, Users, and Organizations, in addition to others. These resource classes provide methods that allow access to their functionality.

In general, most of the methods often require a JSON object that represents the data the resource method needs. Successful requests will generally return a string, JSON object or Stream(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,and the certificate. 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);

Workspaces workspaces = apiSession.GetWorkspacesResource();

LoginResult loginResult =
    apiSession.StartSessionWithServiceAccount(username,
                                              serviceAccount,
                                              expiresInMinutes,
                                              certificate);

ItemListJson<WorkspaceInfoJson> itemListJson = workspaces.listRoomsV30(workspaceTypes);
```

Many domain-specific errors will result in a WebException being returned to the caller. More-generalized C# errors like Exception are also possible.

Examples for using the Workspaces 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 to create a Self-Signed certificate, or you can purchase a commercial certificate from any number of certificate providers.

   For example we can construct and pass the certificate object with the local path to the generated certificate, a password if there is any and set the X509KeyStorageFlags to Exportable or PersistKeySet.

   ```
   X509Certificate2 certificate = new X509Certificate2(path,
                                                       password,
                                                       X509KeyStorageFlags.Exportable |
                                                       X509KeyStorageFlags.PersistKeySet);
   ```

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 accounts** 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 theiremail 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 use the StartSessionWithServiceAccount method on ApiSession to authenticate a session.

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 new ApiSession(serverUrl);
LoginResult loginResult =
    apiSession.StartSessionWithServiceAccount(username,
                                              serviceAccountName,
                                              expiresInMinutes,
                                              certificate);
```

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-----
```

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 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 the LoginResult value is returned from the method. If the user provided a correct set of credentials the StartSessionWithOAuth method returns a "Success" LoginResult.

```
ApiSession apiSession = new ApiSession(serverUrl);
LoginResult loginResult = apiSession.StartSessionWithOAuth(email);
```

7.3 Manually authenticating using OAuth

The basic flow is as follows where WORKSPACES_URL refers to the base URL of the Workspaces server being used:

1. Make an unauthenticated call to *WORKSPACES_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):

   ```
   <WORKSPACES_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).

   ```
   <WORKSPACES_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)
{
  String authToken = String.Empty;

  String serviceAccountName = "com.watchdox.system.xxxx.yyyy";
  int expiresInMinutes = 5;
  X509Certificate2 certificate = new X509Certificate2("myCertFile.crt",
                                                      "certPassword",
                                                      X509KeyStorageFlags.Exportable |
                                                      X509KeyStorageFlags.PersistKeySet);

  ApiSession apiSession = new ApiSession(serverUrl);

  LoginResult loginResult =
    apiSession.StartSessionWithServiceAccount(username,
                                              serviceAccountName,
                                              expiresInMinutes,
                                              certificate);

  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);
  LoginResult loginResult = apiSession.LoadExistingSession(authToken);
}
```

Configuration Settings

The BlackBerry Workspaces SDK has several settings which control the behavior of the SDK. These settings are specified in your applications configuration. If a particular setting is not specified in an application's configuration the SDK will use a default value for that setting.

9.1 Specifying configuation settings

The BlackBerry Workspaces SDK uses the following settings for handling configuration files which will be configured in the App.config of the specific project.

First define a section for BlackBerry.Workspaces.Settings in configSection of the App.config like this:

```
<configSections>
  <section
       name="BlackBerry.Workspaces.Settings"
       type="System.Configuration.ClientSettingsSection, System, Version=4.0.0.0,
             Culture=neutral, PublicKeyToken=b77a5c561934e089"
       allowExeDefinition="MachineToLocalUser"
       requirePermission="false" />
</configSections>
```

Then add the list of settings inside the section defined for BlackBerry.Workspaces.Settings:

```
<BlackBerry.Workspaces.Settings>
  <setting name="DefaultServerURL" serializeAs="String">
    <value>www.watchdox.com</value>
  </setting>
  <setting name="DownloadBufferSize" serializeAs="String">
    <value>4000</value>
  </setting>
  <setting name="DownloadToTmp" serializeAs="String">
    <value>True</value>
  </setting>
</BlackBerry.Workspaces.Settings>
```

9.2 Accessing configuration settings

The following snippet shows how to get or set the settings using BlackBerry.Workspaces.Settings.Portable:

```
//set
BlackBerry.Workspaces.Settings.Portable["DownloadBufferSize"] = 5000;

//get
int bufferSize = BlackBerry.Workspaces.Settings.Portable.GetInt("DownloadBufferSize");

byte[] buffer = new byte[bufferSize];
```

9.3 Available configuration settings

These are the types and default values used in the BlackBerry.Workspaces.Settings.

AutoSignInEmail

used to set or get auto signIn email.

```
Type: string
```

DefaultServerURL

used to set or get auto signIn email.

```
Type: string
Default value: www.watchdox.com
```

DownloadBufferSize

used to set or get the download buffer size.

```
Type: int
Default value: 4000
```

DownloadToTmp

used to set or get whether to download a file to a temporary location.

```
Type: bool
Default value: true
```

MaximumTimeoutForUpload

used to set or get the maximum time out when downloading a file.

```
Type: double
Default value: 6000
```

OAuthRedirectUrl

used to set or get Ouath redirect URL.

```
Type: string
Default value: www.google.com
```

Enable Logging

The Workspaces SDK uses NLog as its underlying logging provider. Additional Information about NLog found at NLog Tutorial

10.1 Configuring Logging

To enable logging in the SDK you need to add to the configuration for your application. First define a *section* for NLog in the *configSection* of your configuration.

```
<configSections>
  <section name="nlog" type="NLog.Config.ConfigSectionHandler, NLog"/>
</configSections>
```

Next, add the targets and rules inside the section defined for NLog. Targets specify where to write log messages, Rules define when to write log messages.

In the example below two targets have been defined: one will send log messages to a file named "workspacessdk.log", the other will write to the console. The two rules specified will cause **Trace** and higher messages to be written to the log file while only **Info** and higher messages will be written to the console.

```
<nlog
     autoReload="true"
     xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <targets>
    <target
         name="logfile"
         xsi:type="File"
         fileName="${basedir}/workspacessdk.log"
         layout="${level} | ${message}"
         keepFileOpen="true" />
    <target
         name="console"
         xsi:type="Console" />
  </targets>

  <rules>
    <logger name="*" minlevel="Trace" writeTo="logfile"/>
    <logger name="*" minlevel="Info" writeTo="console" />
  </rules>
</nlog>
```

10.2 Workspaces logging extensions

The Workspaces SDK provides two extensions to NLog.

- **CustomDbgOutputTarget:** Logs messages using the native OutputDebugString method.
- **ResettableFileTarget:** Provides a File target which will reset or truncate the log file whenever a specified pattern is seen in a log message.

To enable the Workspaces SDK extensions you must add an *extensions* section to the NLog configuration section.

```
<nlog
     autoReload="true"
     xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <extensions>
    <add assembly="WorkspacesSdk" />
  </extensions>
</nlog>
```

The CustomDbgOutputTarget extension extends the NLog TargetWithLayout which allows the *layout* attribute to be specified to define the format of the log messages.

```
<nlog
     autoReload="true"
     xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <extensions>
    <add assembly="WorkspacesSdk" />
  </extensions>

  <targets>
    <target
         name="ods"
         xsi:type="CustomDbgOutput"
         layout="${level} | ${message}"/>
  </targets>

  <rules>
    <logger name="*" minlevel="Trace" writeTo="ods"/>
  </rules>
</nlog>
```

The ResettableFileTarget extension extends the NLog FileTarget. Any attributes allowed on FileTarget maybe used with ResettableFileTarget.

ResettableFileTarget supports an additional attribute, **DeleteToken**, which controls when the log file is reset/truncated. The default value for **DeleteToken** is *~DELETE~LOG~FILE~*. Whenever the value specified by the **DeleteToken** attribute is seen in a log message the current logfile is truncated and any future log messages will be written from the beginning of the log file.

```
<nlog
     autoReload="true"
     xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <extensions>
    <add assembly="WorkspacesSdk" />
  </extensions>

  <targets>
    <target
         name="resetlogfile"
         xsi:type="ResettableFile"
         deleteToken="SomeDeleteToken"
         fileName="${basedir}/workspacessdk.log"
         layout="${message}"
         keepFileOpen="true" />
  </targets>

  <rules>
    <logger name="*" minlevel="Trace" writeTo="resetlogfile"/>
  </rules>
</nlog>
```

Additional Information about NLog found in here NLog Tutorial

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.

12.1 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 certificate to use when signing the data.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 new ApiSession(serverUrl);
LoginResult loginResult =
    apiSession.StartSessionWithServiceAccount(username,
                                              serviceAccount,
                                              expiresInMinutes,
                                              certificate);
```

12.2 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 new ApiSession(serverUrl);
LoginResult loginResult = apiSession.StartSessionWithOAuth(email);
```

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).

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

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

```
            #region devGuide1

            Resource.Workspaces workspaces = apiSession.GetWorkspacesResource();

            // Create the JSON object needed for the method call, and set its values.
            CreateRoomJson createRoomJson = new CreateRoomJson
            {
                Name = name,
                Description = description,
                Administrators = administrators
             };

            // Call the method, and get a JSON object back
            RoomJson roomJson = workspaces.CreateRoomV30(createRoomJson);

            #endregion
```

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

```
#region devGuide2

Resource.Workspaces workspaces = apiSession.GetWorkspacesResource();

// Create a JSON object the represents the new group and set its values
PermittedEntityFromUserJson permittedEntityFromUserJson =
    new PermittedEntityFromUserJson
    {
        Address = groupName,
        EntityType = EntityType.GROUP
    };

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

// The JSON object needed for the resource method call
AddEntityVdrJson addEntityVdrJson = new AddEntityVdrJson
{
    PermittedEntity = permittedEntityFromUserJson,
    NewPermissions = permissionFromUserJson
};

// Make the call to the Rooms resource and return "success"
return workspaces.AddEntityV30(workspaceId, addEntityVdrJson);

#endregion
```

13.3 Add members (users) to group

```
            #region devGuide3

            Resource.Workspaces workspaces = apiSession.GetWorkspacesResource();

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

            // Loop through the List<String> userAddresses
            foreach (string currentAddress in userAddresses)
            {
                PermittedEntityFromUserJson currentEntity = new PermittedEntityFromUserJson
                {
                    Address = currentAddress,
                    EntityType = EntityType.USER
                };

                //make a AddMemberToGroupJson for each user
                AddMemberToGroupJson currentMemberJson = new AddMemberToGroupJson
                {
                    Entity = currentEntity
                };

                memberList.Add(currentMemberJson);
            }

            // Set the group name to be a string and roomId is the integer identifying the room
            AddMembersToGroupWithGroupJson groupMemberJson = new AddMembersToGroupWithGroupJson
            {
                MembersList = memberList,
                RoomId = roomId,
                GroupName = groupName
            };

            // Make the call to the Rooms resource and return "success"
            string result = workspaces.AddMembersToGroupV30(groupMemberJson);

            #endregion
```

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.

14.1 Get list of documents in a Workspace

```
            #region devGuide4

            Resource.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);

            #endregion
```

14.2 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.

```
            #region devGuide5

            PermittedEntityFromUserJson groupPermission = new PermittedEntityFromUserJson
            {
                Address = groupName,
                EntityType = EntityType.GROUP

            };

            List<PermittedEntityFromUserJson> permissionsList =
                new List<PermittedEntityFromUserJson>
                {
                    groupPermission
                };

            // Adjust permissions as needed
            PermissionSetJson permissionSet = new PermissionSetJson
            {
                DownloadOriginal = YesNoDefault.YES,
                DownloadControlled = YesNoDefault.YES,
                Copy = YesNoDefault.YES,
                Edit = YesNoDefault.YES,
                Print = YesNoDefault.YES,
                ProgrammaticAccess = YesNoDefault.YES,
                Spotlight = YesNoDefault.YES,
                Watermark = YesNoDefault.YES
            };

            HashSet<string> documentGuids = new HashSet<string>
            {
                myDoc.Guid
            };

            VdrAddPermissionsJson permissionsJson = new VdrAddPermissionsJson
            {
                PermittedEntities = permissionsList,
                PermissionSet = permissionSet,
                DocumentGuids = documentGuids
            };

            #endregion
```

14.3 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).

```
            #region devGuide6

            // Add permissions
            BulkOperationResultJson result =
                workspaces.AddPermissionsV30(roomId, permissionsJson);

            #endregion
```

Example 4: Upload a file

15.1 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.

```
#region devGuide16

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

// Create a new SubmitDocumentsVdrJson JSON
SubmitDocumentsVdrJson uploadInfo = new SubmitDocumentsVdrJson
{
    OpenForAllRoom = false,
    Recipients = new RoomRecipientsJson
    {
        Groups = groups,
        Domains = domains,
    },
    Folder = folder,
    TagValueList = null,
    DeviceType = DeviceType.SYNC
};

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

#endregion
```

15.2 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.

```
            #region devGuide17

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

            // Create a new SubmitDocumentSdsJson JSON
            SubmitDocumentSdsJson uploadInfo = new SubmitDocumentSdsJson
            {
                // A call to generate a guid for the document and assign it to DocumentGuids
                DocumentGuids = new HashSet<string> { uploadManager.GetNewGuidForDocument()
                },
                // Create a new permission JSON
                Permission = new PermissionFromUserJson
                {
                    Copy = true,
                    Download = true,
                    DownloadOriginal = false,
                    ExpirationDate = DateTime.Now
                },
                UserRecipients = userRecipients,
                ActiveDirectoryGroupsRecipients = ADGroupsRecipients,
                ListRecipients = listRecipients,
                WhoCanView = WhoCanView.RECEIPIENTS_ONLY
            };
            // A call to the UploadDocument
            UploadResult uploadResult =
                uploadManager.UploadDocument(uploadInfo, localPath, null, filename, null);

            #endregion
```

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.

16.1 Using DownloadFileById

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

```
            #region devGuide18

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

            // A call to the DownloadFileById
            downloadManager.DownloadFileById(docId, string.Empty, roomId,
                destinationPath, lastUpdateTime, true, true);

            #endregion
```

16.2 Using DownloadFileByName

This method uses Document name to download the file :

```
            #region devGuide19

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

            // A call to the DownloadFileByName
            downloadManager.DownloadFileByName(roomId, folderPath, docName,
                destinationPath, lastUpdateTime);

            #endregion
```

16.3 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.

```
            #region devGuide20

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

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

            #endregion
```

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.

```
            #region devGuide9

            Resource.Workspaces workspaces = apiSession.GetWorkspacesResource();

            // Create an object to specify the details of what documents to list and how
            // they are returned. A few options are shown here.
            ListDocumentsVdrJson selectionJson = new ListDocumentsVdrJson
            {
                OrderAscending = false,
                FolderPath = "/"
            };

            // Call the list method
            PagingItemListJson<BaseJson> response =
                workspaces.ListDocumentsV30(roomId, selectionJson);

            #endregion
```

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.

```
            #region devGuide10

            // Create the json that indicates the group that has permissions to the document
            PermittedEntityFromUserJson groupPermission = new PermittedEntityFromUserJson
            {
                Address = groupName,
                EntityType = EntityType.GROUP
            };

            List<PermittedEntityFromUserJson> permissionsList =
                new List<PermittedEntityFromUserJson>
                {
                    groupPermission
                };

            // Add the document GUID
            HashSet<string> documentGuids = new HashSet<string>
            {
                myDoc.Guid
            };

            // Create the json for the permissions and set them to NO
            PermissionSetJson permissionSet = new PermissionSetJson
            {
                DownloadOriginal = YesNoDefault.NO,
                DownloadControlled = YesNoDefault.NO,
                Edit = YesNoDefault.NO,
                Print = YesNoDefault.NO,
                Spotlight = YesNoDefault.NO,
                Watermark = YesNoDefault.NO,
                Copy = YesNoDefault.NO,
                ProgrammaticAccess = YesNoDefault.NO,
            };

            VdrEditPermissionsJson editPermissionsJson = new VdrEditPermissionsJson
            {
                PermittedEntities = permissionsList,
                DocumentGuids = documentGuids,
                PermissionSet = permissionSet
            };

            // Make the call and get a BulkOperationResultJson back
            BulkOperationResultJson result =
                workspaces.EditPermissionsV30(roomId, editPermissionsJson);

            #endregion
```

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.

19.1 Get list of workspaces (rooms) for the user

```
            #region devGuide11

            Resource.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(null, true, true, false, false);

            #endregion
```

19.2 Get list of folders in a specific Workspace

```
            #region devGuide12

            Resource.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.SubFolders;

            #endregion
```

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.

```
            #region devGuide13

            Files files = apiSession.GetFilesResource();

            // Create an object to specify the documents activityLog request
            GetDocumentActivityLogRequestJson getDocumentActivityLogRequestJson =
                new GetDocumentActivityLogRequestJson
            {
                // The guid of a document to retrieve activity for
                DocumentGuid = documentGuid,

                // Indicates if only the last action for a user should be retrieved
                LastActionPerUser = false,

                // Indicates the page number to fetch of a multipage response
                PageNumber = 1,

                // The number of items to fetch per page
                PageSize = 100

            };

            // Call the get activity method
            PagingItemListJson<ActivityLogRecordJson> result =
                files.GetActivityLogV30(getDocumentActivityLogRequestJson);

            #endregion
```

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

```
#region devGuide3

Resource.Workspaces workspaces = apiSession.GetWorkspacesResource();

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

// Loop through the List<String> userAddresses
foreach (string currentAddress in userAddresses)
{
    PermittedEntityFromUserJson currentEntity = new PermittedEntityFromUserJson
    {
        Address = currentAddress,
        EntityType = EntityType.USER
    };

    //make a AddMemberToGroupJson for each user
    AddMemberToGroupJson currentMemberJson = new AddMemberToGroupJson
    {
        Entity = currentEntity
    };

    memberList.Add(currentMemberJson);
}

// Set the group name to be a string and roomId is the integer identifying the room
AddMembersToGroupWithGroupJson groupMemberJson = new AddMembersToGroupWithGroupJson
{
    MembersList = memberList,
    RoomId = roomId,
    GroupName = groupName
};

// Make the call to the Rooms resource and return "success"
string result = workspaces.AddMembersToGroupV30(groupMemberJson);

#endregion
```

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.

```
            #region devGuide14

            Resource.Workspaces workspaces = apiSession.GetWorkspacesResource();

            // Create an object to specify the documents to delete and if that will be permanent.
            DeleteDocumentsSelectionVdrJson documentGuidsJson =
                new DeleteDocumentsSelectionVdrJson
            {
                // Set permanent document deletion
                IsPermanent = true,

                // The set of guids for the documents to delete
                DocumentGuids = guidsForDeletion
            };

            // Call the delete method
            BulkOperationResultJson bulkOperationResultJson =
                workspaces.DeleteDocumentsV30(roomId, documentGuidsJson);

            #endregion
```

Example 12: 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.

```
            #region devGuide15

            Files files = apiSession.GetFilesResource();

            // Create an object to specify the documents to delete and if that will be permanent.
            DeleteDocumentsSelectionSdsJson documentGuidsJson =
                new DeleteDocumentsSelectionSdsJson
            {
                // Set permanent document deletion
                IsPermanent = true,

                // The set of guids for the documents to delete
                DocumentGuids = guidsForDeletion
            };

            // Call the delete method
            BulkOperationResultJson bulkOperationResultJson =
                files.DeleteDocumentsV30(documentGuidsJson);

            #endregion
```

Example 13: Handling events

Several objects in the Workspaces .NET SDK generate events when certain things happen such as when a user has successfully been authenticated or when an file upload has completed. The following examples show how to make use of these events.

Classes that generate events:

- ApiSession
- SignInManager
- DownloadManager
- UploadManager

Consult the Workspaces .NET SDK documention for the events generated by each of these classes.

24.1 User authenticated event

In this example the *SignInCompleteEventHandler* method is added to the *SignInCompleted* event of the SignInManager. Once this has been done, any time a user is successfully authenticated on the instance of the ApiSession class the event will be fired and the *SignInCompleteEventHandler* method would be called.

```
        #region devGuide1

        public ApiSession InitializeSession (string workspacesServer, string userEmail)
        {
            // create a new ApiSession object
            ApiSession apiSession = new ApiSession(workspacesServer);

            // get a reference to the SignInManager
            ISignInManager signInMgr = apiSession.SignInManager;

            // register our event handler
            signInMgr.SignInCompleted += SignInCompleteEventHandler;

            // login the user using OAuth
            LoginResult loginResult = apiSession.StartSessionWithOAuth(userEmail);

            // return the ApiSession
            return apiSession;
        }

        private void SignInCompleteEventHandler(object sender, EventArgs eventArgs)
        {
            Console.WriteLine("User signin completed");
        }

        #endregion
```

24.2 Session expired event

Whenever the authentication token for a user expires the Workspaces .NET SDK will attempt to obtain a new token. If the SDK is unable to create a new token it will fire a SessionExpired event. In this example the *SessionExpiredEventHandler* method is added to the ApiSession instance.

```
        #region devGuide2

        public ApiSession InitializeSession(string workspacesServer)
        {
            // create a new ApiSession object
            ApiSession apiSession = new ApiSession(workspacesServer);

            // register our event handler
            apiSession.SessionExpired += SessionExpiredEventHandler;

            // return the ApiSession
            return apiSession;
        }

        private void SessionExpiredEventHandler(object sender)
        {
            Console.WriteLine("Session expired");
        }

        #endregion
```

24.3 Upload progress event

When a file is being uploaded it might be good to provide feedback to a user on how the upload is progressing. By providing an event handler to the UploadManager an application could do just that. This example adds the

OnUploadProgressChangedEventHandler method to the *OnUploadProgressChanged* event of an instance of the UploadManager class. At regular intervals the Workspaces .NET SDK will fire this event as a file is being upload.

```
        #region devGuide3

        public void UploadFile(ApiSession apiSession,
                               SubmitDocumentsVdrJson uploadInfo,
                               int workspaceId,
                               string destinationName,
                               string filename)
        {
            // get the UploadManager
            UploadManager upldMgr = apiSession.GetUploadManager();

            // register our event handler
            upldMgr.OnUploadProgressChanged += OnUploadProgressChangedEventHandler;

            // upload the file
            upldMgr.UploadDocumentToRoom(uploadInfo,
                                         workspaceId,
                                         destinationName,
                                         filename,
                                         null);
        }

        private void OnUploadProgressChangedEventHandler(
                        object sender,
                        IFileRequestData fileRequestData,
                        int progress,
                        UploadStatus status)
        {
            Console.WriteLine("Uploading " + fileRequestData.LocalPath +
                              ", " + progress + " percent complete.");
        }

        #endregion
```

Legal notice

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