BlackBerry Workspaces™

REST Developer's Guide

BlackBerry 2020-11-05

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 API 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 web API 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

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 API design borrows concepts from REST. It consists of a set of HTTP requests for resources using GET and POST methods, sent to the Workspaces server. The request may include parameters in the URL path and/or a message body in JSON format. The following is an example of a generic message:

```
GET https://www.workspaces.com/api/3.0/room/{roomId}/new
Authorization: Bearer <ssid>
Content-type: application/json

{
  path: String
}
```

In the above example, the parameter roomId is included in the URL. An authorization header line follows (see below) and then a header line indicates that the MIME type of the message body is in JSON format. The message body follows, in this example, specifying the path field.

The HTTP Responses from the server include an HTTP status line and, optionally, response data in JSON format

Upload file message format

Files are uploaded in multipart/form-data. The following example shows the format:

```
POST https://www.workspaces.com/api/3.0/documents/{guid}/upload HTTP/1.1
Content-Type: multipart/form-data; boundary=--------------------------114782935826962
Content-Length: 454

----------------------------114782935826962
Content-Disposition: form-data; name="data"; filename="Filename.doc"
Content-Type: application/octet-stream

... File data here in binary ...
----------------------------114782935826962--
```

Response Status and error codes

The Workspaces server returns standard HTTP status codes in response to requests.

HTTP Status	Description
200 OK	The request completed successfully.
201 Created	A request that created a new resource completed, and no response body containing a representation of the new resource is being returned.
202 Accepted	The request has been accepted for processing, but the processing has not completed.
204 No Content	The server fulfilled the request, but does not need to return a response message body.
400 Bad Request	The request could not be processed because it contains missing or invalid information. For example,a validation error on an input field, or a required value is missing.
401 Unauthorized	The authentication credentials included with this request are missing or invalid.
403 Forbidden	The server recognized the requester's credentials, but the requester does not have authorization to perform this request.
404 Not Found	The request specified a URI of a resource that does not exist.

405 Method Not Allowed	The HTTP command specified in the request (DELETE, GET, HEAD, POST, or PUT) is not supported for this request URI.
409 Conflict	A create or update request could not be completed because it would cause a conflict in the current state of the resources supported by the server. For example, an attempt to create a new resource with a unique identifier is already assigned to an existing resource.
500 Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
501 Not Implemented	The server does not currently support the functionality required to fulfill the request.
503 Service Unavailable	The server is currently unable to handle the request due to temporary overloading or maintenance of the server.

Successful requests generally return an HTTP status code of 200 (OK), 201 (Created), or 204 (No Content), to indicate that the requested action has been successfully performed. In addition, successful requests may include a response message body containing a representation of the requested information.

Error responses can be run-time, in response to severe problems with the request message which prevent the Workspaces server from processing it at all (for example, a request on a non-existent workspace), or bulk error responses. These are errors in part of a request (eg. for requests with multiple parts) or errors in requests that can be processed.

Run-time errors will return an HTTP Status Code of type 40x/50x and a run-time JSON error message block in the body of the response message. For example:

```
HTTP/1.1 400
Content-type: application/json

{
  "messages" : [ {
    "action" : "NO_DOCUMENTS_IN_ZIP_OUTPUT",
    "code" : 330,
    "severity" : "ERROR",
    "text" : "No documents were found for adding to zip file "
  } ]
}
```

Explanation of fields:

Action	Error description text
Code	Error code (enum)
Severity	Severity of error (enum)
Text	Description of error (plain text)

Bulk error messages return a code of 200 and a JSON error message block in the response message body. The message may be an aggregated result with a single code summarizing multiple requests. For example:

```
HTTP/1.1 200
Content-type: application/json

{
  "problematicItems" : [ {
    "errors" : [ {
      "errorCode" : 371,
      "isAggregatedMessage" : true,
      "errorArgs" : [ "1" ],
      "errorMessage" : "Attempted to download 1 folders, all of them empty"
    } ]
  } ],
  "fullSuccess" : false,
  "success" : "PARTIAL"
}
```

Alternatively, the message can indicate the result of each request separately. For example:

```
{
  "problematicItems" : [ {
    "itemId" : "aaaa",
    "errors" : [ {
      "errorCode" : 300,
      "isAggregatedMessage" : false,
      "errorArgs" : [],
      "errorMessage" : "Document was not found"
    } ]
  } ],
  "fullSuccess" : false,
  "success" : "PARTIAL"
}
```

Explanation:

errorCode	Error code (enum)
isAggregateMessage	Boolean: true if error relates to a few items (eg on multiple files), or false if each item has its own exception.
errorArgs	Aggregate number of operations
errorMessage	Description of error (plain text)
fullSuccess	Boolean: indicates whether bulk operation was successful (all operations)
Success	Enum: indicates degree of success of all operations if 'fullSuccess' flag is False.

Creating a Session

An application that uses the Workspaces service must be authenticated to start a session. A secure session id (SSID) token is generated and passed to the application during the authentication. It must be used in all requests messages during that session.

There are two methods of authentication:

1. Workspaces email authentication - the application provides an email address (eg. alice@abc.com) and a password in the create resource. If these values are valid, the service returns a secure session id (SSID) token.
2. External authentication - the application authenticates to an external Identity Provider over OAuth2.0 and uses the returned token in a session with Workspaces.

Authentication using Service Accounts

6.1 Overview

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.

The steps for creating and using service accounts are as follows:

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 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-----
```

6.3 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-----
```

6.4 Creating an authentication token

To create an autentication token 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 how long the signed token will be valid for. How an application determines or aquires a users email address would be up to the applicaiton. The issuer is identifier assigned when the service account was created in the WatchDox 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 timestamp is a number of seconds since January 1, 1970, 00:00:00 GMT.

The first step is to create a string in which the user, issuer and expires values are formatted as they would be in an URL with each of the values URL encoded using UTF-8.

If we assume the following:

user

user1@mycompanydomain.biz

issuer

com.watchdox.system.fadd.3015

expires

1472688269

The URL formatted string would look like this:

```
user=user1%40mycompanydomain.biz&issuer=com.watchdox.system.fadd.3015&expires=1472688269&
```

Next the string must be signed using the private key from the certificate. Exactly how this is done will vary depending you the language being used. If Java is being used with a Java KeyStore the process would involve getting the private key from the keystore, and using the Java Security APIs to sign the string with the private key.

After the string has been signed, the binary result should be Base64 encoded. The original string should then be concatenated with the Base64 encoded, signed value using a colon to seperate the two.

Note: This example wraps the value for readabilty. The actual value would not have any line breaks

```
user=user1%40mycompanydomain.biz&issuer=com.watchdox.system.fadd.3015&expires=1472688269& \
: \
gYaao7kCrmihuQ/fiYzlPZO1gaiekxdKGEoUPFrjYQhtX+rDw22wD6aYDUvI3uPVe6/e8NkOVmoBIZhgI7OyMz \
0aDXraqW5gP6GzmhLOvS4lD2iFo+/3+UXxkuE8lcGDkOjjcAbnVP4hPtdiLwRk3x+0BZx6yS15ktNjj20mS4b/mk \
O3ukP7wZfnoFA8Lf/vVbfGb72CkeYRZVwkhoeeoeb5dQTOWoyP8Fsv3inCKShnsX1cKVCoJJ49sJoKYZ4EQCw3uy \
6hZm3Tv3sD8aWBxMygeh318D0042f2zb6CNTaEd4TSQT1UqeqiMy2Yw9lhGKeJarnSXmbx3fY6gYwTyQ==
```

The resulting value value is then used as the authentication token in the HTTP Authoriziation header.

```
Authentication: Bearer user=user1%40mycompanydomain.biz&issuer=com.watchdox.system.fadd.3 ...
```

Authentication using OAuth

7.1 Overview

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.

7.2 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.

7.3 Steps in 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.

Examples

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

In cases where the message structure is complex, only the fields pertinent to the example are shown, and the rest is represented by "...".

Note that the term room is interchangeable with workspace.

Example 1: Connect and authenticate a user

This example shows how to connect a user to the service by using OAuth to login and create a session. This requires registration as an OAuth provider to obtain a clientId and client secret that will be passed to Workspaces in order to obtain a Workspaces authentication token.

The server returns an access token that must be included in subsequent requests in the session.

9.1 Determine the URI to be used

Make an unauthenticated request to get the authorization URI's. The URIs of interest are authorizationUri and accessTokenUri.

Request

```
GET /api/3.0/authentication/parameters
```

Response

```
{
  "isOauth" : true,
  "authorizationUri" : "<WDX_SERVER_URL>/ngdox/oauth/up/signin/",
  "accessTokenUri" : "<WDX_SERVER_URL>/api/3.0/authentication/token",
  ...
}
```

Parameters

```
isOauth - indicates whether or not the server supports OAuth authentication
authorizationUri - identifies the URL a user should be directed to to signin
accessTokenUri - identifies the URL to be used in requesting an Access token or
  Refresh token.
```

9.2 Direct the user to the Workspaces signin page

Direct the user to the Workspaces authenticationUri (e.g. via a redirect in your own web app), passing the proper parameters. One of these params is the redirect URI in your app to which the user should be sent after authenticating with Workspaces. The request will look something like this (wrapped across lines here, but one single line when used):

Request

```
GET /ngdox/oauth/up/signin/?response_type=code& client_id=<CLIENT_ID>&locale=en_US&
redirect_uri=<REDIRECT_URI>
```

Parameters

```
response_type - should always be "code"
client_id - would be the identfier registered with the Workspaces server for the application.
locale - indicates the language that should be used for the signin page. If the Workspaces
  server does not support the requested locale English will be used.
redirect_uri - indicates where the user should be redirected after successfully
  authenticating. The URI must have been previously registered with the Workspaces server and
  associated with the client_id being used.
```

Response

```
HTTP 302 redirect to
  https://<REDIRECT_URI>/?code=219e5a32-d74f-473a-91a4-fd74f95e091c&locale=en-us
```

9.3 Retrieve the Access token

Using the *code* parameter returned in the redirect URL, make a request to fetch the access token from the server.

Request

```
GET /api/3.0/authentication/token/?client_id=<CLIENT_ID>&client_secret=<CLIENT_SECRET>&
grant_type=authorization_code&redirect_URI=<REDIRECT_URI>&
code=219e5a32-d74f-473a-91a4-fd74f95e091c
```

Parameters

```
client_id -  would be the identfier registered with the Workspaces server for the application.
client_secret - is the secret string associated with the client id.
grant_type - would be "authorization_code" when requesting an Access token using a code
  received in a redirect URL.
redirect_uri - must match the redirect uri provided in the initial request to the sign in
  page.
```

Response

```
{
   error: String,
   error_description: String,
   token_type: String,
   refresh_token: String,
   expires_in: Long,
   access_token: String
}
```

Parameters

```
If there was an error the (error) and (error_description) fields.
If the request was successful the (access_token) field contains the token to be used in
subsequent API requests.
The (refresh_token) would be used to obtain an new access token when the current token
becomes invalid.
The (expires_in) field indicates when the current access token will expire.
```

That access token is used in subequent requests in an HTTP authorization header like this;

Authorization: Bearer expires=1473288317&user=.....

Example 2: Add users to a workspace

This example shows how to add a user to a workspace. It requires three separate requests: create a workspace, create a new room group, and add members (users) to the group. Different permissions are required for each step.

10.1 Create a workspace

(can only be done by an Organization Administrator)

Request

```
POST /api/3.0/rooms/create
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  name: "myroom",
  description: "this is my room",
  administrators: "admin@abc.com"
}
```

Parameters

```
Room name ("myroom")
ssid ("xxxxxx")
description and administrator name
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  id: "37",
  name: "myroom",
  description: "this is my room"
  ...
}
```

Parameters

```
Room id (37)
```

10.2 Create new room group

(can only be done by a Workspace Administrator)

Request

```
POST /api/3.0/rooms/37/entities/add
Authorization: Bearer <xxxxx>
Content-type: application/json

{
  Address: "mygroup",
  permittedEntity: {
    address: "mygroup",
    entityType:GROUP,
    role: VISITORS,
    newPermissions: {
      downloadOriginal: False,
      download: False,
      copy: True,
      print: True,
      edit: True,
      neverExpires: True,
      progAccess: Boolean
    }
  }
}
```

Parameters

```
  Room id (37),
  ssid, from previous request
  Group name ("mygroup")
  Permissions
```

Response

```
HTTP/1.1 200 OK
```

10.3 Add members (users) to group

Request

```
POST /api/3.0/rooms/37/groups/mygroup/members/add
Authorization: Bearer <xxxxxxx>
Content-type: application/json

{
  membersList: List( {
    entity: {
      address: "alice@abc.com",
      entityType: USER
    },
    entity: {
      address: "bob@abc.com",
      entityType: USER
    }
  } )
}
```

Parameters

```
Room id (37),
ssid,
group ("mygroup"),
users to add (alice@abc.com, bob@abc.com)
```

Response

```
HTTP/1.1 200 OK
```

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

This example shows how to allow a group of users to access a document. It assumes a workspace (room 37) and a group ("mygroup") exists - see Example 2. The document name in this example is "mydocument.doc". This example requires two separate requests: retrieve a list of files in the workspace with no filtering, and add permissions for the group to access the file.

11.1 Retrieve the list of documents in a workspace

Request

```
POST /api/3.0/rooms/37/documents/list
Authorization: Bearer <xxxxxx>

{
  pageSize: 20,
  pageNumber: 1,
  folderPath: "\afolder",
  documentOrder: DOCUMENT_NAME,
  orderAscending: True,
  folders: True,
  documentFilter: ALL,
  filterScope: ENTIRE_ROOM,
  noTags: True
}
```

Parameters

```
Room id (37),
ssid token
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  total: 1,
  items: List( {
    id: Integer,
    guid: "4bd675d8-1904-4288-9a82-89bdb1c56dfc",
    ...
    filename: "mydocument.doc",
    ...
    room: String,
    folder: "\afolder"
    ...
  } )
}
```

Parameters

```
List of documents in the workspace (room), including guids for each file.
In this case, the file in question is in the returned list, and its guid
is included (4bd675d8-1904-4288-9a82-89bdb1c56dfc).
```

11.2 Add permissions to the file

Add permissions for the group "mygroup" to access the file, referenced by its guid

Request

```
POST /api/3.0/rooms/37/documents/permissions/add
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  Address: "mygroup",
  permittedEntities: List( {
    entityType: GROUP
  } ),
  permissionSet: {
    downloadOriginal: Yes,
    ...
  },
  documentGuids: "4bd675d8-1904-4288-9a82-89bdb1c56dfc",
  ...
}
```

Parameters

```
Room id (37),
ssid token,
permissions list (download original permitted),
document guid ("4bd675d8-1904-4288-9a82-89bdb1c56dfc")
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  fullSuccess: Yes,
  success: Full
}
```

Example 4: Upload a file to workspace

This example shows how to upload a file to a workspace assuming that a workspace has already been created. There are three steps: create a guid for the new file, upload the file, and submit the groups (recipients) that are allowed to access the file.

12.1 Create a guid for the document in the workspace

Request

```
POST /api/3.0/rooms/37/documents/create
Authorization: Bearer <xxxxxx>
```

Parameters

```
Room id (37),
ssid token (xxxxxx)
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  guid: "7bc67511-164b-5744-2382-8a7db1c56dfc"
}
```

Parameters

```
The new guid (7bc67511-164b-5744-2382-8a7db1c56dfc) is returned.
```

12.2 Upload the file to the workspace

Request

```
POST /api/3.0/rooms/37/documents/7bc67511-164b-5744-2382-8a7db1c56dfc/upload
Authorization: Bearer <xxxxxx>
Content-type: multipart/form-data
```

Parameters

```
Room id (37),
document guid (7bc67511-164b-5744-2382-8a7db1c56dfc),
ssid token (xxxxxx)

File data is uploaded to the Workspaces server as HTTP multipart form-data.
```

Response

```
HTTP/1.1 200 OK
```

12.3 Submit the list of groups (recipients) that are allowed to access the file

Request

```
POST /api/3.0/rooms/37/documents/submit
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  documentGuids: "7bc67511-164b-5744-2382-8a7db1c56dfc",
  recipients: {
    groups: "mygroup",
    ...
  },
  ...
}
```

Parameters

```
The group (mygroup) is associated with the file. The permissions are also set but
this is not illustrated in the example.
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  total: 1,
  items: List( {
    id: Integer,
    guid: "7bc67511-164b-5744-2382-8a7db1c56dfc",
    ...
  })
}
```

Example 5: Send a file via Workspaces

This example shows how to send a file to a recipient. The guid for the file is created, the file is uploaded to Workspaces, and a link is sent to the email recipients.

13.1 Create the file guid

Request

```
POST /api/3.0/documents/create
Authorization: Bearer <xxxxxx>
```

Parameters

```
Session token (xxxxxx)
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  guid: "2dc68811-374b-52c4-2382-897db7d56dfc"
}
```

Parameters

```
Guid for new document (2dc68811-374b-52c4-2382-897db7d56dfc)
```

13.2 Upload the file

Request

```
POST /api/3.0/documents/2dc68811-374b-52c4-2382-897db7d56dfc/upload
Authorization: Bearer <xxxxx>
Content-type: multipart/form-data
```

Parameters

```
Document guid (2dc68811-374b-52c4-2382-897db7d56dfc),
session token (xxxxxx),
document data (multipart form-data).
```

Response

```
HTTP/1.1 200 OK
```

13.3 Submit request to send link to recipient (bob@abc.com)

Request

```
POST /api/3.0/documents/submit
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  userRecipients: "bob@abc.com",
  permission: {
    print: Boolean,
    edit: Boolean,
    ...
  },
  documentGuids: "2dc68811-374b-52c4-2382-897db7d56dfc"
}
```

Parameters

```
Document guid (2dc68811-374b-52c4-2382-897db7d56dfc),
recipients' names (bob@abc.com),
permissions list
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  total: 1,
  items: List( {
    id: 1,
    guid: "2dc68811-374b-52c4-2382-897db7d56dfc",
    ...
  }),
  ...
}
```

Example 6: Enumerate files in a workspace

This example shows how to request a list of all files in a workspace that a user has permission to access.

Request

```
POST /api/3.0/rooms/37/documents/list
Authorization: Bearer <xxxxxx>

{
  ...
}
```

Parameters

```
Room id (37),
session token (xxxxxx).

Additional Parameters in the {...} block specify filters and ordering for the enumeration.
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  total: 2,
  items: List( {
    id: Integer,
    guid: "23a78931-3749-5184-2ca2-907dc8d59dfc",
    filename: "document1.doc",
    permissionsJson: {
      print: True,
      edit: True,
      spotlight: False,
      watermark: True,
      neverExpires: True,
      ...
    },
    documentName:"document1.doc"
  },
  {
    guid: "27678251-3843-5184-2ca3-2c7dc8d59dfc",
    filename: "document2.doc",
    permissionsJson: {
      print: False,
      edit: True,
      spotlight: False,
      watermark: True,
      ...
    },
    documentName:"document2.doc"
  } )
}
```

Parameters

```
In this example, two documents are returned in the enumerated list (document1.doc
and document2.doc).
```

Example 7: Change file permissions

This example shows how to change access permissions for files in a workspace. In this example, file access permissions are changed for all files that a group (“mygroup”) has access rights to. The permissions are set to enable editing and disable printing, spotlight view, and watermarks. Other permissions remain the same.

Request

```
POST /api/3.0/rooms/37/documents/permissions/edit
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  permittedEntities: List( {
    address: "mygroup",
    entityType: GROUP
  }),
  permissionSet: {
    print: "No",
    edit: "Yes",
    spotlight: "No",
    watermark: "No"
  },
  ...
}
```

Parameters

```
Room id (37),
session token (xxxxxx),
Permissions (Edit), groups to apply this to ("mygroup")
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  fullSuccess: "Yes",
  success: "Full"
}
```

The response indicates that the request was completed successfully.

Example 8: Enumerate folders and workspaces

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

16.1 Retrieve the list of workspaces (rooms) for the user.

Request

```
GET /api/3.0/rooms
Authorization: Bearer <xxxxxx>
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  total: 1,
  items: List( {
    id: 37,
    name: "Room1",
    description: "Primary room",
    ...
  })
}
```

Parameters

```
Number of rooms (1),
list of workspaces, including room 37.
```

16.2 Get list of folders in workspace 37

Request

```
GET/api/3.0/rooms/37/folders
Authorization: Bearer <xxxxxx>
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  id: Integer,
  hasSubfolders: "Yes",
  room: "37",
  folder: "\String,
  subFolders: List(
    "subfolder1",
    "subfolder2"
  ),
  fullPath: String "\myworkspace\myfolder\"
}
```

Parameters

```
Name of the folder ("myfolder"),
name of the subfolders ("subfolder1","subfolder2"),
path of folder ("\myworkspace\myfolder")
```

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

This example shows how to obtain a file guid for a named file and retrieve the activity log for this file using the guid. There are two requests: retrieve the guid for a named file, and retrieve the activity log for the file by referencing the guid.

17.1 Retrieve the guid for named file, where the name is "myFile.doc"

Request

```
POST /api/3.0/rooms/37/documents/guid
Authorization: Bearer <xxxxxx>

{
  folderPath: "\afolder",
  documentName: "myFile.doc"
}
```

Parameters

```
Room id (37),
Session token (xxxxxx),
document path ("\afolder") and name ("myFile.doc")
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  guid: "23173251-344c-2b84-63a3-19adc8d59dfc"
}
```

Parameters

```
The guid for the document (23173251-344c-2b84-63a3-19adc8d59dfc)
```

17.2 Obtain the activity log for a file by referencing its guid

Request

```
POST /api/3.0/documents/activityLog
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  documentGuid: "23173251-344c-2b84-63a3-19adc8d59dfc"
}
```

Parameters

```
document guid (23173251-344c-2b84-63a3-19adc8d59dfc),
session token (xxxxxx)
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  total: 2,
  items: List( {
    email: "bob@abc.com",
    activity: "Downloaded original document", details: "-",
    time: "Mon Jun 4, 2012 8:25:20 PST",
    location: "Palo Alt,CA, United States",
    ip: "79.179.169.154",
    device: "8CC133AA"
  },
  {
    email: "tom@abc.com",
    activity: "Opened document",
    details: "On iPad / iPhone",
    time: "Wed Jun 4, 2012 11:03:290 PST",
    location: "Palo Alt,CA, United States",
    ip: "66.201.44.15",
    device: "A3B7A1DE"
   } )
}
```

Parameters

```
The response shows that two users accessed the file. In this example, bob@abc.com downloaded
the file and tom@abc.com opened the file on his iPad or iPhone.
```

Example 10: Add a user to a room group

In this example, the user tom@abc.com is added to the group "mygroup".

Request

```
POST /api/3.0/rooms/37/members/add
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  newMembers: {
    permittedEntity: {
      address: "tom@abc.com",
      entityType: USER
    },
    group: {
      address: "mygroup",
      entityType: GROUP
    }
  }
}
```

Parameters

```
Room id (37),
session token (xxxxxx),
the user that you want to add (tom@abc.com),
the group that you want to add the user to ("mygroup")
```

Response

```
HTTP/1.1 200 OK
```

Example 11: Delete a file from a workspace

This example shows how to delete a file from a workspace. It requires two steps: retrieve the guid for the file that you want to delete, and request to delete the file from its room.

19.1 Retrieve the guid for the file that you want to delete:

Request

```
POST /api/3.0/rooms/37/documents/guid
Authorization: Bearer <xxxxxx>

{
  folderPath: "\myfolder",
  documentName: "myFile.doc"
}
```

Parameters

```
Room id (37),
Session token (xxxxxx),
document path ("\myfolder") and name ("myFile.doc")
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  guid: "23173251-344c-2b84-63a3-19adc8d59dfc"
}
```

Parameters

```
The guid for the document (23173251-344c-2b84-63a3-19adc8d59dfc)
```

19.2 Request to delete the file

Request

```
POST /api/3.0/rooms/37/documents/delete
Authorization: Bearer <xxxxxx>
Content-type: application/json
{
  folderPaths: "\myfolder",
  documentGuids: "23173251-344c-2b84-63a3-19adc8d59dfc"
}
```

Parameters

```
Room id (37),
session token (xxxxxx),
document path ("\myfolder"),
document guid (23173251-344c-2b84-63a3-19adc8d59dfc).
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  fullSuccess: "Yes",
   success: "Full"
}
```

The response indicates that the request was completed successfully, and the file was deleted from the workspace. The document is moved to the Recycle Bin, where it can be recovered if necessary.

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

This example shows how to delete a file sent via Workspaces. It requires two steps: retrieve the guid for the file that you want to delete, and request to delete the file from its room.

20.1 Obtain the file guid by searching the database for the file name

Request

```
POST /api/3.0/documents/search
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  searchString: "myfile.doc"
}
```

Parameters

```
Session token (xxxxxx),
file name to search for ("myfile.doc")
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  total: 1,
  items: {
    guid: "23173251-344c-2b84-63a3-19adc8d59dfc",
    filename: "myfile.doc"
  }
}
```

Parameters

```
Number of search results (1),
document guid (23173251-344c-2b84-63a3-19adc8d59dfc),
file name ("myfile.doc")
```

20.2 Request to delete the file

Request

```
POST /api/3.0/documents/delete
Authorization: Bearer <xxxxxx>
Content-type: application/json

{
  documentGuids: "23173251-344c-2b84-63a3-19adc8d59dfc"
}
```

Parameters

```
Session token (xxxxxx),
document guid (23173251-344c-2b84-63a3-19adc8d59dfc)
and flag indicating not to force delete.
```

Response

```
HTTP/1.1 200 OK
Content-type: application/json

{
  fullSuccess: "Yes",
  success: "Full"
}
```

The response indicates that the request was completed successfully.

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