CloudBackend

SDK API wide overview

version 1.3

2022-01-04

CloudBackend - Copyright © 2020-2022

Document overview

The following API document will give an overview on how to use the different backend calls in the SDK of CloudBackend (CBE).

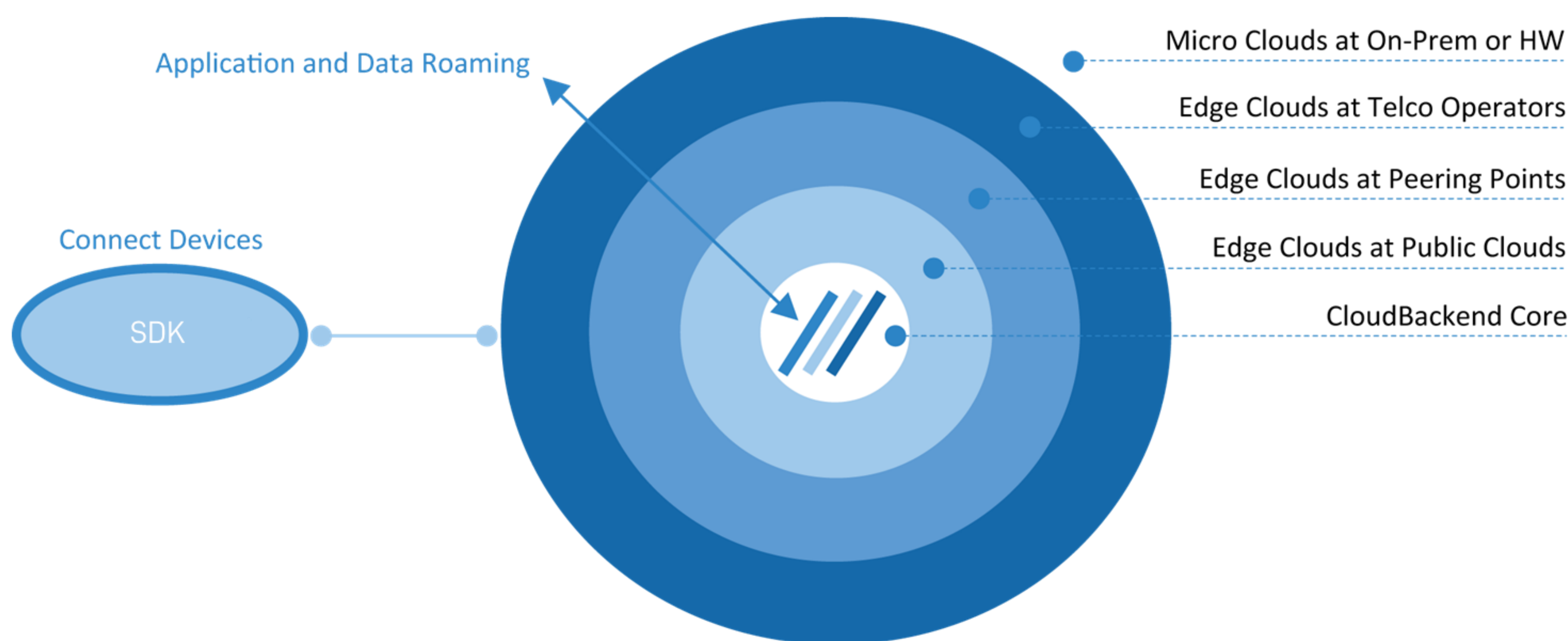
Select the tab of your language of interest to view the code and added usage examples. This will then select what is printed.

This document should be used in conjunction with the SDK tutorial and QUERY user guide.

Conventions

Describing text is printed in font Calibri.

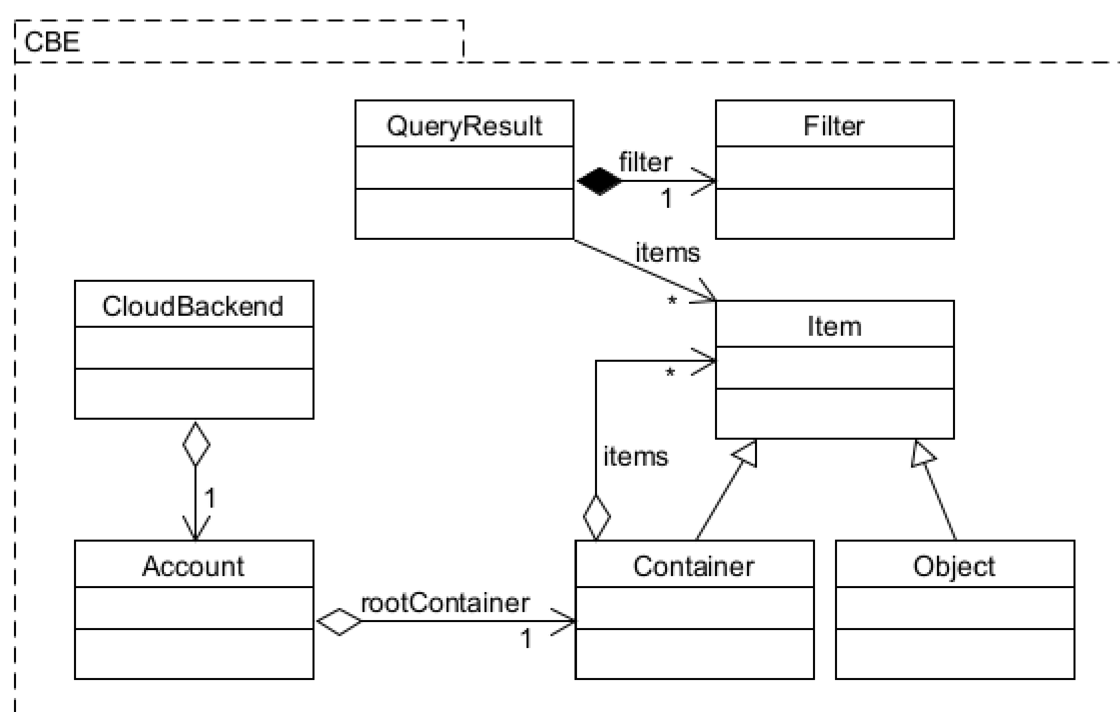
The relation of the CloudBackend layers is illustrated as



CloudBackend system

Classes

The classes used when writing programs using the CloudBackend SDK API.



CloudBackend classes

CloudBackend - Copyright © 2020-2022

CloudBackend

CloudBackend is an EntryPoint into the SDK. Creating an instance of CloudBackend will be the first thing you add. To start using the SDK a developer needs to call **logIn** to get an instance of CloudBackend.

C++

```
CBE::CloudBackendPtr cloudBackend = CBE::CloudBackend::logIn(Username, Password, Tenant, accountDelegate);
```

Once signed in the developer will be able to access the users account, containers, objects and other functionality. The variable has now been assigned the root container and can be used to begin accessing containers and objects in the account. See the account section for more information on how to access the root container.

C++

```
CBE::AccountPtr account = cloudBackend->account();
CBE::ContainerPtr rootContainer = account->rootContainer();
```

When you have signed in and have your account you will be able to access your root container as shown under the Account. On the root container you will be able to perform queries to see what is in the account and add data to the account as shown under Containers. Querying will return Items that can be either Containers or Objects as detailed under Items.

For a detailed walkthrough of how to get started using the CloudBackend SDK see the SDK tutorial.

Language specific implementation details

C++

```
This is the base of the SDK with all its features. It requires C++11 or later.
```

CloudBackend - Copyright © 2020-2022

Getting started working with CloudBackend

Declaration:

C++

```
static CloudBackendPtr logIn(const std::string& username, const std::string& password, const std::string& tenant, CBE:
```

Description:

This should be the first function called when using CloudBackend and should give access to all the other functionality within the SDK as well as signing in the user.

Parameters:

`username`	is the username of the user being signed in
`password`	is the password for the user being signed in
`tenant`	is the identifier for the tenant
`delegate`	uses callbacks **onLogin()** or **onError()** in class AccountEventProtocol on this completion.

Usage:

C++

```
CBE::CloudBackendPtr cloudBackend  = CBE::CloudBackend::logIn(testUsername, testPassword, testSource, accountDelegate)
```

Declaration:

C++

```
CBE::AccountPtr account();
```

Description:

The account is the entry point into the users account and container structure through the root container. Additionally it provides information on the account of the user.

Usage:

C++

```
CBE::AccountPtr account = cloudBackend->account();
```

In addition to signing in the user, the CloudBackend will provide access to the account, a root container, and some other functionality.
See account section for more information.

CloudBackend - Copyright © 2020-2022

Advanced Querying

Declaration:

C++

```
virtual void query(uint64_t containerId, CBE::ItemDelegatePtr delegate);
```

Description:

This is a call to get a list of items in the container. It is intended to be used as a shortcut when the developer knows what container they want to access. To access the root container the developer should use the account. Implement callback onQueryLoaded in class **ItemEventProtocol** to receive the callback.

Parameters:

`containerId`	is the numeric id for the container you want to query. This would need to be retrieved from the parent container. See Account for root container id.
`delegate`	uses callback **onQueryLoaded()** or **onLoadError()** in class **ItemEventProtocol**

Usage:

C++

```
CBE::ItemDelegatePtr itemDelegate = getPtr();
cloudbackend->query(containerId, itemDelegate);
```

Declaration:

C++

```
virtual void query(CBE::container_id_t containerId, CBE::Filter filter, CBE::ItemDelegatePtr delegate);
```

Description:

Call to get a list of items in the folder using a filter. This is intended to be used as a shortcut when the developer knows what container they want to access. To access the root container the developer should use the account . Implement **onQueryLoaded** from ItemEventProtocol to receive the callback. Requires that you are already loged in.

Parameters:

`containerId`	is the numeric id for the container you want to query. This would need to be retrieved from the parent container. For root container, see Account.
`filter`	can be used to set parameters for the query. See Filter for details.
`delegate`	uses callback **onQueryLoaded()** or **onLoadError()** in class **ItemEventProtocol**.

Usage:

C++

```
CBE::ItemDelegatePtr itemDelegate = getPtr();
CBE::Filter filter;
filter.setAscending(false);
filter.setDataType(CBE::ItemType::Container);
cloudbackend->query(containerId, filter, itemDelegate);
```

For more details on advanced queries and **setQuery(std::string)** *and the other Advanced query filters, see the QUERY user guide document.*

CloudBackend - Copyright © 2020-2022

Listeners

Declaration:

C++

```
void addListener(CBE::ItemDelegatePtr delegate);
```

Description:

Adds a listner that will recieve updates as changes occur on the account.

Note! Accounts are always updated automatically, but the notification of the update is provided by the listener functionality. **removeListener** should always be called when you stop using the delegate.

Parameters:

`delegate` is a class implementation of ItemEventProtocol class.

Usage:

C++

```
cloudbackend->addListener(itemDelegate);
```

Declaration:

C++

```
void removeListener(CBE::ItemDelegatePtr delegate);
```

Description:

Removes the listener that is passed in.

Parameters:

`delegate` is the previously passed in class object ItemEventProtocol class.

Usage:

C++

```
cloudbackend->removeListener(itemDelegate);
```

CloudBackend - Copyright © 2020-2022

CloudBackend class Utility

Declaration:

C++

```
static CBE::ContainerPtr castContainer(CBE::ItemPtr item);
```

Description:

This casts an item to a container.

Parameters:

`item` is the **item** you want to cast to a container. This is intended as a helper function.

Usage:

C++

```
CBE::ContainerPtr container =  cloudbackend->castContainer(item);
```

Declaration:

C++

```
static CBE::ObjectPtr castObject(CBE::ItemPtr item);
```

Description:

This casts an item to an object.

Parameters:

`item` is the **item** you want to cast to an object. This is intended as a helper function.

Usage:

C++

```
CBE::ObjectPtr object =  cloudbackend->castObject(item);
```

Declaration:

C++

```
std::string version();
```

Description:

This returns the version of the SDK and can be used for debugging.

Usage:

C++

```
std::cout << "CloudBackend SDK version: " << cloudBackend->version() << std::endl;
```

CloudBackend - Copyright © 2020-2022

class Account

- Account currently offers access to a root container and has additional information on the account like the users first and last name.
- The **rootContainer** is a useful starting point that can then be used to query, create and perform other functions.
- The account is returned by the instance of CloudBackend you received when calling **logIn()**.
- The rootContainer for the account is a good place for you to start querying or creating data.

C++

```
CBE::AccountPtr account = cloudBackend->account()
```

It holds useful information such as:

Root container

Declaration:

C++

```
CBE::ContainerPtr rootContainer account->rootContainer()
```

Description:

This returns the rootContainer for the account.

Usage:

C++

```
CBE::ContainerPtr rootContainer account->rootContainer();
```

Root container id

Declaration:

C++

```
CBE::ContainerId id1
```

Description:

This returns the rootContainerId for the account.

Usage:

C++

```
CBE::ContainerId id1 account->rootContainer()->id();
```

CloudBackend - Copyright © 2020-2022

Account Data

Declaration:

C++

```
CBE::user_id_t userId1 account->userId();
```

Description:

This returns the account id of the user.

Declaration:

C++

```
std::string username1 account->username();
```

Description:

This returns the username of the user.

Declaration:

C++

```
std::string password1 account->password();
```

Description:

This references the password of the account.

Declaration:

C++

```
std::string source1 account->source();
```

Description:

This returns the source identifier for the tenant of the account.

Declaration:

C++

```
std::string firstName1 account->firstName();
```

Description:

This returns the name of the user.

Declaration:

C++

```
std::string lastName1 account->lastName();
```

Description:

This returns the surname of the user.

Declaration:

C++

```
CBE::ContainerPtr rootContainer1 account->rootContainer();
```

Description:

This returns the **rootContainer** for the account.

CloudBackend - Copyright © 2020-2022

class Container

Containers provides a convenient way to organize data in your account and can be used to hold objects and other sub-containers in a hiearchical tree.

To start working with data in an account one can begin with the rootcontainer located on the account. Here you can start creating objects, uploading, querying container contents, and creating sub-containers. E.g.

C++

```
CBE::ContainerPtr rootContainer cloudBackend->account()->rootContainer();
rootContainer->query(itemDelegate);
rootContainer->createObject(mapOfKeyValues);
```

This will receive callbacks from the library. It is wrapping a class where you have implemented the desired functions from ItemEventProtocol.

Queries

Declaration:

C++

```
virtual void query(CBE::ItemDelegatePtr delegate);
```

Description:

Call to get a list of items in the container.

Parameters:

`delegate` uses callback **onQueryLoaded()** or **onLoadError()** in class ItemEventProtocol.

Declaration:

C++

```
virtual void query(CBE::Filter filter, CBE::ItemDelegatePtr delegate);
```

Description:

Call to get a filtered list of items in the container.

Parameters:

`filter` can be used to set parameters for the query. See Filter for details.
`delegate` uses callback **onQueryLoaded()** or **onLoadError()** in class ItemEventProtocol.

CloudBackend - Copyright © 2020-2022

Adding data

Declaration:

C++

```
virtual CBE::ObjectPtr createObject(std::string name, CBE::ItemDelegatePtr delegate, std::map<std::string, SDK_tuple<s
```

Description:

Creates an object with optional indexed tags for faster searches.

Parameters:

`name`	is the title name of the object.
`delegate`	uses callback **onObjectAdded()** or **onItemError()** in class ItemEventProtocol.
`metadata`	is a map called Obj_KV_Map with the tag (key), the value of that tag and if it is indexed or not.
`[indexed]`	indexed is optional
	for more info look at starting section or look for details in com.cbe-source folder and Obj_KV_Map.java

Declaration:

C++

```
virtual CBE::ObjectPtr upload(const std::string& name, const std::string& path, CBE::TransferUploadDelegatePtr delegat
```

Description:

Uploads to container an object with the given file name and path. The object is instantly returned with a temporary id. Once the response from the server is called back, the object gets updated with the correct unique object id.

Parameters:

`name`	is the name of the object.
`path`	is the path to the object.
`delegate`	uses callback **onObjectUploaded()**, **onChunkSent()** or **onObjectUploadFailed()** in class TransportEventProtocol.

Declaration:

C++

```
virtual CBE::ObjectPtr upload(const std::string& name, uint64_t length, char* byteData, CBE::TransferUploadDelegatePtr
```

Description:

Uploads to the container an object with the given binary data held in memory. The object is instantly returned with a temporary id. Once the response from the server is called back, the object gets updated with the correct unique object id.

Parameters:

`name`	is the name of the object.
`length`	is the length of the object in bytes.
`byteData`	the byte array containing the data.
`delegate`	uses callback **onObjectUploaded()**, **onChunkSent()** or **onObjectUploadFailed()** in class TransportEventProtocol.

CloudBackend - Copyright © 2020-2022

Organization

Declaration:

C++

```
virtual ContainerPtr create(const std::string& name, CBE::ItemDelegatePtr delegate);
```

Description:

This creates a sub-container inside this container, to be used for adding objects.

Parameters:

`name`	is the name for the container.
`delegate`	uses callback **onContainerAdded()** or **onItemError()** in class ItemEventProtocol.

Declaration:

C++

```
virtual void move(CBE::container_id_t destinationId, CBE::ItemDelegatePtr delegate);
```

Description:

Move is used to move container with its content to user specified location e.g. id of other container or to root container.

Parameters:

`destinationId`	is the id of the container to which it should be moved to.
`delegate`	uses callback **onContainerMoved()** or **onItemError()** in class ItemEventProtocol.

Declaration:

C++

```
virtual void remove(CBE::ItemDelegatePtr delegate);
```

Description:

This will delete the container and all its content.

Parameters:

`delegate`	uses callback **onContainerRemoved()** or **onItemError()** in class ItemEventProtocol.

Declaration:

C++

```
virtual void rename(const std::string& name, CBE::ItemDelegatePtr delegate);
```

Description:

This changes the name of the container.

Parameters:

`name`	is the new name.
`delegate`	uses callback **onContainerRenamed()** or **onItemError()** in class ItemEventProtocol.

CloudBackend - Copyright © 2020-2022

ACL

ACLs can be set on either individual objects or for complete container tree structures containing both sub containers and objects.

The data type for setting ACL for objects and containers are defined in **object** and **container**.

Below is a list of permissions with some comments:

```
 0 = no permissions, this alternative should be set to be sure a user can not access anything and should be called before or right after unsharing a Container / container structure or Object.
 1 = read only, enables the user to query and download.
 2 = write only, enables the user to upload to a shared container and rename a shared object or container. However if there are multiple sub containers in the structure the user will not be able to see this since they can not query.
 3 = read/write, combination of 1 and 2 and can be used for creating or uploading objects to any sub container in a container tree and rename/download objects.
 4 = delete, this would most efficiently be used if you share temporarily an object (picture for instance) that the user can remove once seen.
 5 = read/delete, this allows a user to remove containers and objects in a share.
 6 = write/delete, this is necessary to be able to move objects and containers within the shared container.
 7 = read/write/delete, enables rename/upload/download/move/create and remove.
 8 = changeACL only, enables a user to set new acls on a container/object. When doing that remember to include the original owner otherwise ownership will change.
 9 = read/changeACL.
10 = write/changeACL.
11 = read/write/changeACL.
12 = delete/changeACL.
13 = read/delete/changeACL.
14 = write/delete/changeACL.
15 = all permissions meaning read/write/delete/changeACL.
The value is calculated as the sum of changeACL (8) + delete (4) + write (2) + read (1).
```

Declaration:

C++

```
virtual void setACL(std::map<CBE::user_id_t, CBE::permission_status_t> toUserPermissions, CBE::ShareDelegatePtr delega
```

Description:

Set the Access control list of the container. For containers set does set the whole container tree, with all its sub items as well. Remember this is set and not update so everytime you set, all userids that should be there must be included.

Parameters:

`toUserPermissions`	is a class implementing javas AbstractMap where the key is the userId and the value is the permission you want to set. Multiple users can be set through one call to setACL.
`delegate`	is a shared pointer to the class implementing **ShareDelegate**, and is a shared pointer of the template class ShareEventProtocol.

Usage:

C++

```
std::map<CBE::user_id_t,CBE::permission_status_t> userIdPermissionsMap;
userIdPermissionsMap.insert(std::pair<CBE::user_id_t,CBE::permission_status_t>(account2->userId(),3));
userIdPermissionsMap.insert(std::pair<CBE::user_id_t,CBE::permission_status_t>(account3->userId(),5));
userIdPermissionsMap.insert(std::pair<CBE::user_id_t,CBE::permission_status_t>(account4->userId(),7));
container->setACL(userIdPermissionsMap, shareDelegate);
```

CloudBackend - Copyright © 2020-2022

Declaration:

C++

```
virtual void getACL(CBE::ShareDelegatePtr delegate);
```

Description:

Get the ACL (Access Control List) of a container/object.

Parameters:

`delegate`	is a shared pointer to the class implementing **ShareDelegate**, and is a shared pointer of the template class ShareEventProtocol.

Sharing

At present Sharing the container gives the user read permissions for the container and all its sub-items.
Note! This might change in the future.

Declaration:

C++

```
virtual void share(user_id_t toUserGroup, std::string description, CBE::ShareDelegatePtr delegate);
```

Description:

Share a container/object with another user/group. This provides the user the ability to access what has been shared to them via the **listAvailableShares** command, see class ShareManager. ACL should have been set prior to calling this. To allow users to view and change shared information see ACLs.

Parameters:

`toUserGroup`	user id or group id to share to.
`description`	name identifying this specific share between you and the user/group.
`delegate`	is a shared pointer to the class implementing **ShareDelegate**.

Declaration:

C++

```
virtual void unShare(uint64_t shareId, CBE::ShareDelegatePtr delegate);
```

Description:

Remove a previously given share of the container to a specific shareId. Each share is unique between user/group and the one sharing. This is represented with a unique share id.

Parameters:

`shareId`	the unique id for a share between the owner and other user/group.
`delegate`	is a shared pointer to the class implementing **ShareDelegate**.

CloudBackend - Copyright © 2020-2022

class ShareManager

This provides calls to see what you have shared or has been shared to you. Shares are given by a user to other users. The shared container/object must have an ACL specifying the rights given to the specific user.

Declaration:

C++

```
virtual void listAvailableShares(CBE::ShareDelegatePtr delegate);
```

Description:

Lists the shares that have been shared to you. This will give you information similar to a query but with specific share information.

Parameters:

`delegate` is a callback class implementing the ShareEventProtocol class.

Declaration:

C++

```
virtual void listMyShares(CBE::ShareDelegatePtr delegate);
```

Description:

Lists shares that have been shared by you to others. This will give you information similar to a query but with specific share information.

Parameters:

`delegate` is a callback class implementing the ShareEventProtocol class.

CloudBackend - Copyright © 2020-2022

class Item

Item is the base class for both containers and objects. It contains data that can be found on both and allows them to be presented together in a **queryResult**.

Query results returns a list of items. Item can be cast to Objects or Containers by calling

C++

```
CBE::ContainerPtr item = CBE::CloudBackend::castContainer(item);
or
std::static_pointer_cast<CBE::Object>(item);
```

Item data

Declaration:

C++

```
virtual CBE::item_id_t id() const;
```

Description:

This returns an items id.

Declaration:

C++

```
virtual CBE::container_id_t parentId() const;
```

Description:

This returns the id of the Items parent.

Declaration:

C++

```
virtual std::string name() const;
```

Description:

This returns the name.

Declaration:

C++

```
virtual std::string path() const;
```

Description:

This returns the path to the item. It may not return anything if it has not been set.

Declaration:

C++

```
virtual CBE::user_id_t ownerId() const;
```

Description:

This returns the owners id.

CloudBackend - Copyright © 2020-2022

Declaration:

C++

```
virtual std::string username() const;
```

Description:

This returns the username of the Containers owner.

Declaration:

C++

```
virtual CBE::date_t created() const;
```

Description:

This returns the creation date in Unix time.

Declaration:

C++

```
virtual CBE::date_t updated() const;
```

Description:

This returns the updated date and time in Unix time.

Declaration:

C++

```
virtual CBE::date_t deleted() const;
```

Description:

This returns the deleted date in Unix time.

Declaration:

C++

```
virtual CBE::item_t type() const;
```

Description:

This specifies if the item is a Object (=4) or Container (=8).

Declaration:

C++

```
virtual uint64_t length() const;
```

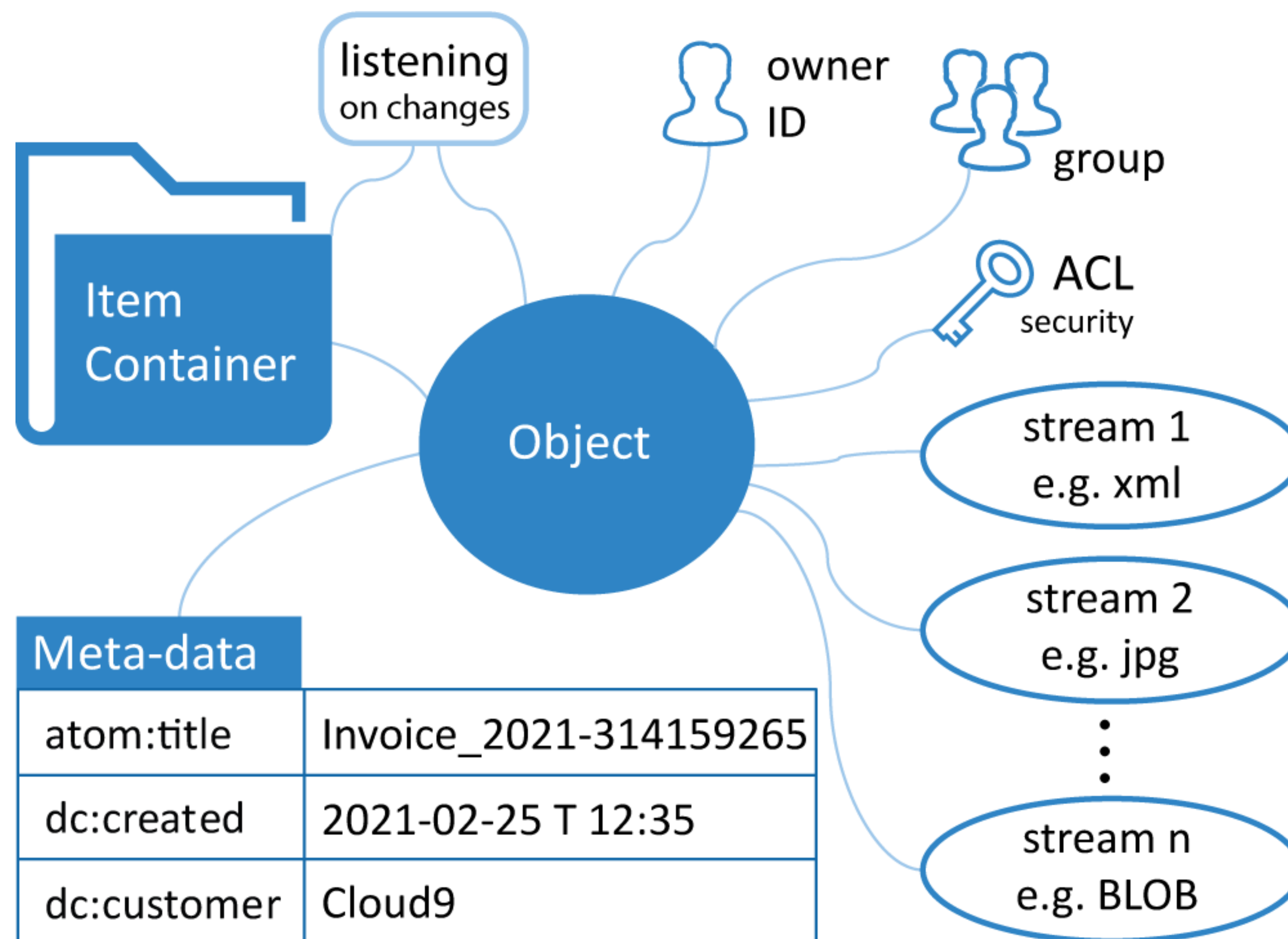
Description:

This returns the byte size of an object (i.e. file size).

CloudBackend - Copyright © 2020-2022

class Object

Objects are a representation of data stored in CloudBackend. They can either contain the complete data or represent data stored and be used to access that data. Object requests that involve server communication take an implementation of either **ItemEventProtocol** class, **TransferEventProtocol** class or **ShareEventProtocol** class as one of the parameters. How to create a protocol is discussed in the protocol section.



Object features example

CloudBackend - Copyright © 2020-2022

Downloading

Declaration:

C++

```
virtual void download(const std::string& path, CBE::TransferDownloadDelegatePtr delegate);
```

Description:

This downloads the object with path from a container to local file store.

Parameters:

`path`	to where it will be downloaded.
`delegate`	is a callback class implementing the TransportEventProtocol class.

Declaration:

C++

```
virtual void download(CBE::TransferDownloadDelegatePtr delegate);
```

Description:

This download the object with binary data from the cloud and passes it to the delegate. This is data is on the heap and you are responsible for calling delete on it, though we may change it to a shared pointer in the future.

Parameters:

`delegate`	is a callback class implementing the TransportEventProtocol class.

CloudBackend - Copyright © 2020-2022

Modifying

Declaration:

C++

```
virtual void move(CBE::container_id_t destinationContainerId, CBE::ItemDelegatePtr delegate);
```

Description:

Move object to another container.

Parameters:

`destinationContainerId`	is the container Id it will be moved to.
`delegate`	is a callback class implementing the ItemEventProtocol class.

Declaration:

C++

```
virtual void rename(const std::string& name, CBE::ItemDelegatePtr delegate);
```

Description:

This changes the name of the object.

Parameters:

`name`	is the name of the object.
`delegate`	is a callback class implementing the ItemEventProtocol class.

Declaration:

C++

```
virtual void remove(CBE::ItemDelegatePtr delegate);
```

Description:

This will delete the object from the cloud container.

Parameters:

`delegate`	is a callback class implementing the ItemEventProtocol class.

Declaration:

C++

```
virtual void updateKeyValues(CBE::ItemDelegatePtr delegate, std::map<std::string,  SDK_tuple<std::string, bool>> metac
```

Description:

Adds keyValue data to the existing object, if data has the same name it will be overwritten, otherwise it will add to the existing keyValue on the object.

Parameters:

`delegate`	is a callback class implementing the ItemEventProtocol class.
`metadata`	is a map with the tag (key), the value of that tag and if it is indexed or not.
`[indexed]`	indexed is optional

CloudBackend - Copyright © 2020-2022

Object data

Declaration:

C++

```
virtual std::string getMimeType() const;
```

Description:

This returns the mime type of the object e.g. xml/text or jpg etc.

Declaration:

C++

```
virtual std::map< std::string, SDK_tuple<std::string, bool> > keyValues();
```

Description:

This returns all the keyValues.

Object Streams

Declaration:

C++

```
virtual void downloadStream(const std::string& path, uint64_t streamId, CBE::TransferDownloadDelegatePtr delegate);
```

Description:

This downloads the stream # with stream id, to a path of your choice. To get which streams are added on a object use getStream.

Parameters:

`path`	is the path you want to download the stream to.
`streamId`	specifies which stream you want by first calling getStream and then choose which one to download.
`delegate`	is a callback class implementing the ItemEventProtocol class.

Declaration:

C++

```
virtual std::vector<CBE::Stream> getStreams(CBE::ItemDelegatePtr delegate);
```

Description:

This returns the streams attached to the Object, use object.streams() to view. Then use the Streams data to put into the downloadStream(..) call above.

CloudBackend - Copyright © 2020-2022

class Query Result

A QueryResult contains a list of items that meet query or search criteria that the user establishes. The simplest query is the contents of a container. For information regarding using the Items you get, see the Item section. More complex queries can be performed on individual containers or the entire account and require the objects match user defined criteria from object names to meta data. Meta data is matching pairs of data attached to an Object like **artist** and "Nirvana" or **latitude** and "58.41". More complex queries require passing in a filter with information on what you want to query for. See the Filter section.

For more details on advanced queries see the QUERY user guide document.

Query data

Declaration:

C++

```
virtual std::vector<CBE::ItemPtr> getItemsSnapshot();
```

Description:

Returns a copy of a vector containing the items for the queryResult. The queryResult will update when new data comes in but the copy will not. If iterating make sure to create a variable for a local copy.

Usage:

C++

```
std::vector<CBE::ItemPtr> itemSnapshot = queryResult ->getItemsSnapshot();
for (std::vector<CBE::ItemPtr>::iterator it = itemSnapshot.begin(); it != itemSnapshot.end(); ++it) {
    std::cout << (*it)->name() << std::endl;
}
```

Declaration:

C++

```
virtual uint64_t itemsLoaded();
```

Description:

This tells how many items are in the queryResult.

Declaration:

C++

```
virtual uint64_t totalCount();
```

Description:

This tells how many total items in the cloud match the query result. This may be more than loaded.

Declaration:

C++

```
bool bypassCache;
```

Description:

This was added to allow users to bypass cached data when pulling data stored on other users accounts as notifications are currently not implemented for this and it will not update automatically.

CloudBackend - Copyright © 2020-2022

class Filter

A Filter is a tool for performing more advanced queries on an account. They allow the specification of a wide range of criteria from a specific range of items to meta data matching. See the SDK **Filter.java** file and the QUERY user guide for more details.

Declaration:

C++

```
CBE::Filter filter;
```

Description:

Filter can be used to see what has been loaded like containerIds, matching meta data, offsets and additional data.

Usage:

C++

```
CBE::Filter filter;
filter.setAscending(false);
filter.setCount(50);
filter.setQuery("type:image/png");
```

CloudBackend - Copyright © 2020-2022

Protocols

To manage asynchronous communication we define a set of callbacks in protocol functions. You will create a class that inherits a protocol and implements the functions for the callbacks you wish to receive from that protocol. See the SDK **include/CBE/Protocols** header directory for more details.

The callbacks should come in a separate thread from the one they were called in. Multple treads can run simultaneously to not block. Multiple calls on the same object may be queue requests if they could be affected by a previous request. As a last measure calls can timeout if they are taking too long to complete.

C++

```
#include "CBE/Protocols/AccountEventProtocol.h"

class AccountEventImplementation : virtual public CBE::AccountEventProtocol {
  public:
    virtual void onLogin(uint32_t atState, CBE::CloudBackendPtr cloudbackend);
    virtual void onError(uint64_t operationId, persistence_t failеAtState, uint32_t code, std::string reason, std::str
}
```

AccountEventProtocol

This provides callbacks that will be used when you call **logIn()** and have inherited **AccountEventProtocol**.

- **onLogin()** should implement code to run: after the login has been successful.
- **onError()** — if the login has failed.

OnError is only ever triggered in response to a call initiated by the current SDK. It will be in a different thread than it is called from.
onLogout() and onCreated() are not used.

See also:

C++

```
See the SDK include/CBE/Protocols/AccountEventProtocol.h header file for more details.
```

CloudBackend - Copyright © 2020-2022

ItemEventProtocol

ItemEventProtocol provides callbacks that will be used when you perform actions on a Container or Object. Additionally it can be used to receive callbacks for **query()**. Finally, if you **addListner()** ItemEvents that trigger on the server, including creating an item will callback to the listener you added. It should be inherited by a class you want to receive these callbacks and you should implement any callbacks you wish to trigger.

- **onObjectAdded()** should implement code to run: after an object has been added.
- **onMetadataAdded()** — after metadata has been added.
- **onObjectMoved()** — after an object has been moved.
- **onObjectRemoved()** — after an object has been removed.
- **onObjectRenamed()** — after an object has been renamed.
- **onObjectUpdated()** — after an object has been updated.
- **onStreamsLoaded()** — after an object has been updated with a stream.
- **onContainerAdded()** — after a container has been added.
- **onContainerMoved()** — after a container has been moved.
- **onContainerRemoved()** — after a container has been removed.
- **onContainerRenamed()** — after a container has been renamed.
- **onContainerRestored()** — after a container has been restored after being deleted.
- **onQueryLoaded()** — after a query was made.
- **onLoadError()** should implement code to run: if a query fails, e.g. a filter requesting a container or object that does not exists.
- **onItemError()** — if an error regarding an item occurred, e.g. create rename, move, remove.

See also:

> **C++**
>
> `See the SDK` **`include/CBE/Protocols/ItemEventProtocol.h`** `header file for more details.`

CloudBackend - Copyright © 2020-2022

ShareEventProtocol

This provides callbacks that will be used when you call **listAvailableShares()** or **listMyShares()** and have inherited **ShareEventProtocol**.

- **onListAvailableShares()** should implement code to run: after a query for available shares that have been shared to you.
- **onListMyShares()** — after a query for shares that you have shared to other accounts or groups.
- **onContainerACLAdded()** — after an Access Control List has been added to a container.
- **onObjectACLAdded()** — after an Access Control List has been added to an object.
- **onContainerAclLoaded()** — after an Access Control List for a container has been loaded.
- **onObjectAclLoaded()** — after an Access Control List for an object has been loaded.
- **onContainerShared()** — after a container and its content has been shared by you to somebody.
- **onContainerUnShared()** — after a container and its content has been unshared by you.
- **onObjectShared()** — after an object has been shared by you to somebody.
- **onObjectUnShared()** — after an object has been unshared by you.
- **onShareError()** should implement code to run: if an error regarding a share has occured.
- **onACLError()** — if a getACL or setACL has failed or is forbidden.

See also:

> **C++**
>
> See the SDK **include/CBE/Protocols/ShareEventProtocol.h** header file for more details.

TransferEventProtocol

TransferEventProtocol is used for uploading and downloading files to containers.

- **onObjectUploaded()** should implement code to run: after a file has been uploaded.
- **onChunkSent()** — after a file chunk has been uploaded.
- **onObjectUploadFailed()** — if there was an error with the upload.
- **onChunkReceived()** should implement code to run: after a file chunk has been recieved.
- **onObjectDownloaded()** — after a file has been downloaded.
- **onObjectBinaryDownloaded()** — after a text file has been loaded on the memory heap.
  Note! *This is downloaded as byte[] data and you are responsable for the management of your heap memory.*
- **onObjectDownloadFailed()** — if there was an error with the download.

See also:

> **C++**
>
> See the SDK **include/CBE/Protocols/TransferEventProtocol.h** header file for more details.

CloudBackend - Copyright © 2020-2022

Synchronous vs. Asyncronous

Many calls to the CBE SDK return a synchronous and asynchronous response. This leads to a decision in how you wish to work based on priorities. All remote communication within the CBE takes place asynchronously. However updates are made to the local cache of data on client requests. When the remote request completes a callback to a delegate you have implemented.

Working synchronously allows one to immediately act on data they have sent to the server. This however means that it is possible someone elsewhere will be looking at different data than you are. Additionally if a request fails, requests dependent on the success of that request that appear complete will be rolled back.

Working asynchronously means that you wait until you receive a response in your delegate to make a follow up request. Working this way guarantees everything you are presenting to your client has been propagated to the edge and will be what is retrieved if another client communicates with the edge. Additionally you will not have requests you believe are complete rolled back.

The CloudBackend Synchronous area and naming conventions may evolve in the future. Please, check back in coming versions of this document.

CloudBackend - Copyright © 2020-2022