CloudBackend

SDK tutorial

version 1.4.6

2022-04-12

Document objective

The following SDK Tutorial document will give an overview on how to use the backend calls in the SDK of CloudBackend (CBE). This is done by listing code examples. The C++ example is used in the descriptions here. A similar example is available in the Java language.

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

Preparation

For this you need the following installed:

- Ubuntu 20.04 LTS Linux operating system

Download the SDK package provided by CloudBackend:

- SDK - Software Development Kit (zip file)

Optional download:

- Tutorial repository

Installation

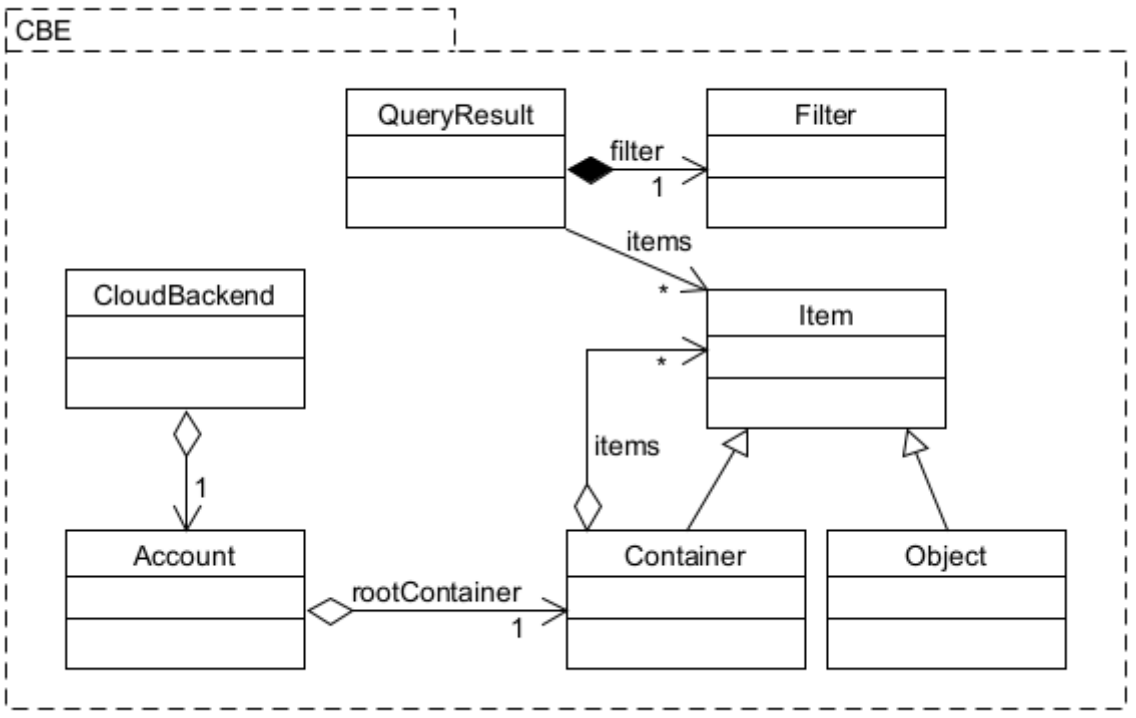
From a terminal window; unpack the package file.

```
sudo apt install g++                                # if not already installed
cd                                                  # the home directory or equivalent
mkdir cbe                                           # if not already created
cd cbe                                              # replace with your project directory
unzip ~/Downloads/CloudBackendSDK-main.zip          # replace with the zip file name previously
downloaded
mv CloudBackendSDK-main 1.4.6                       # release number or another name of your choice
ln -s 1.4.6 current                                 # create symbolic link to the current version of
the SDK
cd current                                          # the name previously given this release's
directory
cd C++/lib/Linux_x86/                               # SDK C++ lib
unzip libcb_sdk.zip                                 # unpack the libcb_sdk.a
cd ../../Examples                                   # Example Code directory
cd Simple                                           # Simple code example directory
sh compile.sh                                       # compile the test program
sh run.sh                                           # run the installation test program and then enter
your name
Name for a new Company Container:  Adam
New container created: Adam
program complete

cd ..                                               # ~/cbe/current/C++/Examples
mkdir -p Tutorial                                   # if not already created
cd Tutorial                                         # put exercise files here
~/cbe/current/C++/Examples/Tutorial
```

Classes

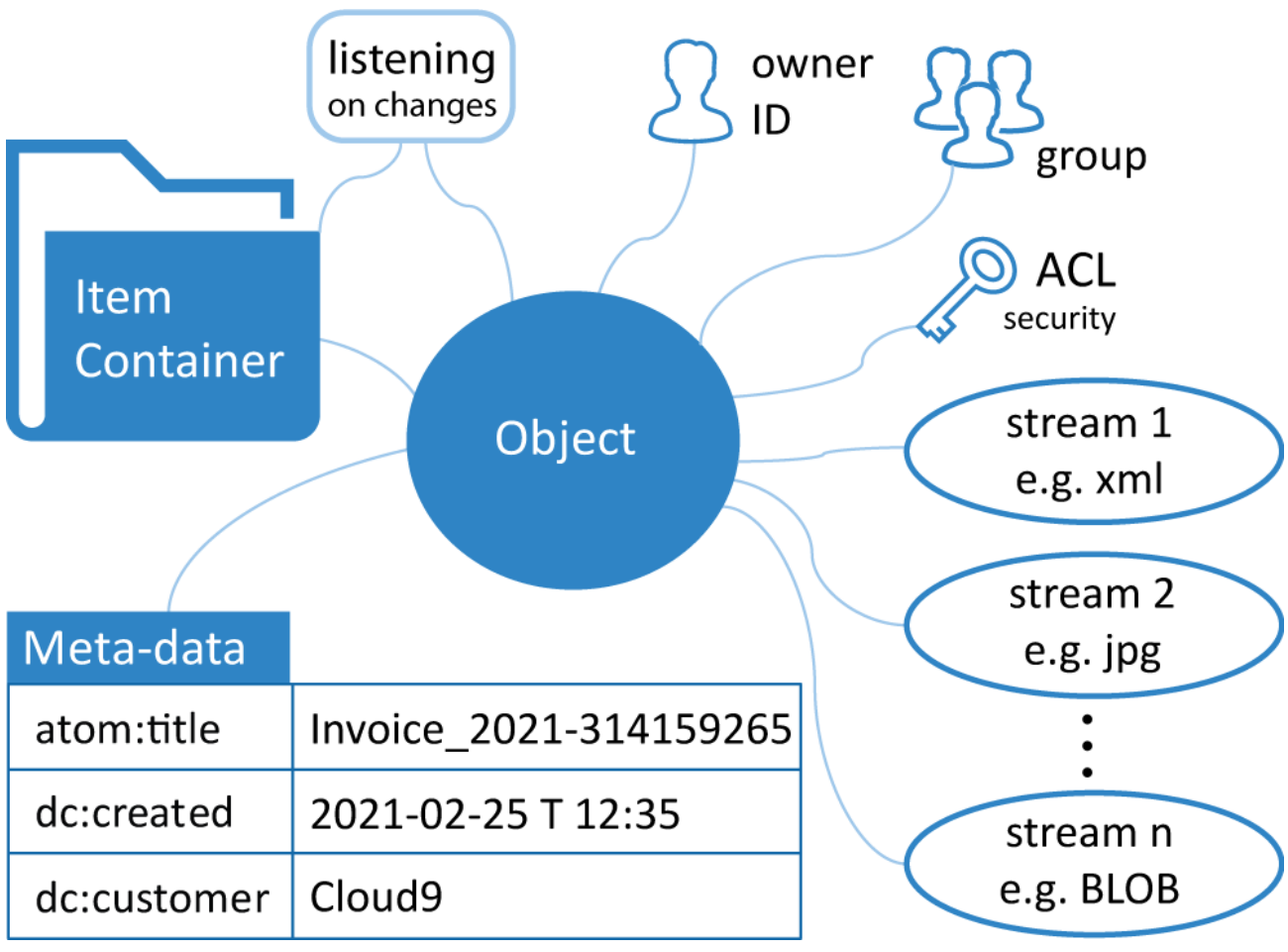
The classes available when writing programs using the CloudBackend SDK API.



CloudBackend classes

Objects

The object features.



Object description

To use the Tutorial

In this tutorial you will need a CloudBackend account, the include files and a copy of the SDK binary called **libcb_sdk.a** which is a static library.

The tutorial will focus on teaching 4 simple methods that are needed to get started with the basics of using the SDK. Remember this is not a course in c++.

The tutorial is structured in 3 exercises: 1st is to login to your account, 2nd is to create a container in the root container and the 3rd exercise is to query the account and create an object on the backend service.

At the end of this document there is a link to download the Tutorial source code.

Exercise 1

To start off we need a main loop that keeps our main thread alive and second we need a class that orchestrates in what order we do API requests. This can be done in two ways; either just one big class and a .h and .cpp file with everything in it: API requests, callbacks and all the logic you want to add. See the SDK included Example Simple. Or, as in this Tutorial, we will have callbacks and logic separated to illustrate how to build with the library by only expanding with what you need. The SDK has as of now 4 different template / protocol classes that implements corresponding delegates / smart pointers which are used for callbacks from the cross API layer to notify when a request is done on the backend and also updated on the cloudbackend objects' local cache.

As a start login we need to add the following source code files in a project folder:

- Main.cpp
- Logic.h
- Logic1.cpp
- AccountEventProtocol.h
- AccountEventProtocol.cpp
- compile.sh
- run.sh

In **Main.cpp** add the following:

```
#include <stdio.h>
#include <iostream>
#include <chrono>
#include <thread>
#include "Logic.h"

Logic* logic = new Logic();

void waitUntilFinished() {
  while (!logic->finished) {
   std::this_thread::sleep_for(std::chrono::seconds(1));
 }
}

int main(void) {
  printf("Hello! Main program started\n");
  logic->start();
  waitUntilFinished();
  delete logic;
  return 0;
}
```

Next we want to define the class Logic in which we have the cloudbackend object and will have the entire API requests of the program.

Create file **Logic.h** and add the following:

```
#ifndef INCLUDE_CBE_SOLUTIONLOGICBEODY_H_
#define INCLUDE_CBE_SOLUTIONLOGICBEODY_H_
#include <iostream>
#include <mutex>
#include "CBE/CloudBackend.h"
#include "CBE/Types.h"

class Logic {
 public:
  void start();
  void logic();
  void programFinished();
  bool finished{};
  CBE::ContainerPtr rootContainer{};
  void saveQueryResultContinue(CBE::QueryResultPtr qR);

private:
 int logicInstances = 0;
 int step = 1;
 std::recursive_mutex logicMutex{};
 CBE::CloudBackendPtr cloudBackend{};
 CBE::QueryResultPtr qResult{};

 // Exercise 2
 // Exercise 3

 // Generic functions
 static const bool  noBoolDefaultVal;
 static bool inquireBool(const std::string& prompt,
                         const bool&        defaultVal = noBoolDefaultVal);

 static const int noIntDefaultVal;
 static int inquireInt(const std::string& prompt,
                       const int&         defaultVal = noIntDefaultVal);

  static const std::string noStringDefaultVal;
  static std::string inquireString(const std::string& prompt,
                                   const std::string& defaultVal =
                                                        noStringDefaultVal);

  static std::string trimString(const std::string& str);

  static const char* itemTypeString(CBE::item_t itemType);
  static void printItem(const CBE::Item& item, bool printParentId = false);

  static std::string containerName(CBE::ContainerPtr  container,
                                   bool               temporary = false);
  static std::string objectName(CBE::ObjectPtr  object,
                                bool            temporary = false);
};
#endif
```

In **Main.cpp** we now know what `Logic* logic = new Logic()` means. As we see there are 3 function declarations and 4 member variables. Start will be the entry point to our program where we log in, a call to logic will be a call to a switch / case statement where steps are used to choose case. The bool finished and function **programFinished()** are used to kill the main thread. Next step is to implement some of the methods provided in the .h-files.

Create **Logic1.cpp** and add this:

```
#include "Logic.h"

#include "AccountEventProtocol.h"

#include <algorithm>  // std::find_if_not
#include <cstring>
#include <cctype>     // std:isspace, std::tolower, std::toupper
#include <ios>        // std::left
#include <iomanip>    // std::setw
#include <sstream>    // std::ostringstream

/* Here we will logIn() */
void Logic::start() {
  const bool login = inquireBool("Do you want to login", true /* defaultVal*/);
  if (!login) {
    programFinished();
    return;
  }
  const auto username = inquireString("Type username", "githubtester2");
  const auto password = inquireString("Type password");
  const auto tenant =   inquireString("Enter  tenant", "cbe_githubtesters");

  CBE::AccountDelegatePtr accountDelegate =
                                  std::make_shared<AccountEventProtocol>(this);
  cloudBackend = CBE::CloudBackend::logIn(username, password, tenant,
                                          accountDelegate);
}

void Logic::programFinished() {
  std::cout << "program finished." << std::endl;
  finished = true;
}

void Logic::saveQueryResultContinue(CBE::QueryResultPtr qR) {
  std::cout << "continue program" << std::endl;
  qResult = qR;
  logic();
}

/* Exercise 2 */

/* Exercise 3 */

/** Logic Exercise */
void Logic::logic(int thisInstance) {

  std::unique_lock<std::recursive_mutex> lock{logicMutex};
```

```
bool continueToNextStep;
++logicInstances;
const int thisInstance = ++logicInstances;
std::cout << "logic instance: " << thisInstance << std::endl;
do {
  continueToNextStep = false;
  std::cout << "step: " << step << std::endl;
  switch (step) {
    /* Exercise 2 */

    /* Exercise 3 */

    /* the end */
    default: {
      std::cout << "Exercise completed!" << std::endl;
      programFinished();
      break;
    }
  }
  } while(continueToNextStep);
  std::cout << "end of do-while loop;"
            << " instance: " << thisInstance
            << " next step: " << step
            << std::endl;
}


/*----- generic functions -----*/
/*-- generic input functions --*/
const bool  Logic::noBoolDefaultVal{}; // = false
bool Logic::inquireBool(const std::string& prompt,
                        const bool&        defaultVal) {
  const bool hasDefaultVal = (&defaultVal != &noBoolDefaultVal);
  constexpr char  falseChar = 'n';
  const char      actualFalseStr =
      (hasDefaultVal && !defaultVal)? std::toupper(falseChar) : falseChar;
  constexpr char trueChar =  'y';
  const char      actualTrueChar =
      (hasDefaultVal && defaultVal)?  std::toupper(trueChar) : trueChar;
  while (true) {
    std::cout << prompt << " (" << actualTrueChar << '/' << actualFalseStr
              << ")? ";
    std::string answer;
    std::getline(std::cin, answer);
    if (answer.empty()) {
      if (hasDefaultVal) {
        return defaultVal;
      }
      continue;
    }
    switch (std::tolower(answer[0])) {
    case trueChar:  return true;
    case falseChar: return false;
    default: continue;
    }
  }
}
```

```
const int Logic::noIntDefaultVal{}; // = 0
int Logic::inquireInt(const std::string& prompt,
                      const int&         defaultVal) {

  const bool hasDefaultVal = (&defaultVal != &noIntDefaultVal);
  while (true) {
    std::cout << prompt
              << (hasDefaultVal? (" (Default value is '" +
                                  std::to_string(defaultVal) + "')") : "")
              << "? ";
    std::string answer;
    std::getline(std::cin, answer);
    if (answer.empty()) {
      if (hasDefaultVal) {
        return defaultVal;
      }
      continue;
    }
    try {
      const int intVal = std::stoi(answer);
      return intVal;
    } catch (std::exception& e) {
      std::cout << "Failed to convert answer \"" << answer << "\" to an integer."
                << "Got exception with what()=\"" << e.what() << "\"" << std::endl;
    }
  }
}

const std::string Logic::noStringDefaultVal{}; // ""
std::string Logic::inquireString(const std::string&  prompt,
                                 const std::string&  defaultVal) {
  const bool hasDefaultVal = (&defaultVal != &noStringDefaultVal);
  while(true) {
    std::cout << prompt
              << (hasDefaultVal? (" (Default value is \"" + defaultVal + "\")")
                               : "") << "? ";
    std::string answer;
    std::getline(std::cin, answer);
    if (answer.empty() && hasDefaultVal) {
      answer = defaultVal;
    }
    const auto /* std::string */ trimmedAnswer = trimString(answer);
    if (trimmedAnswer.empty()) {
      std::cout << "Your input \"" << answer
                << "\" is not valid; please, input a response." << std::endl;
    } else {
      return trimmedAnswer;
    }
  }
}

std::string Logic::trimString(const std::string& str) {
   const auto /* std::string::const_iterator */ noSpaceFront =
       std::find_if_not(str.cbegin() /* first */, str.cend() /* last */,
                        [](int ch){
                          return std::isspace(ch);
```

```
                        } /* q */);
   const auto noSpaceBack =
       std::find_if_not(str.crbegin(),
                        std::string::const_reverse_iterator(noSpaceFront),
                        [](int ch){
                          return std::isspace(ch);
                        }).base();

   return std::string(noSpaceFront, noSpaceBack);
}

/*-- generic print functions --*/
const char* Logic::itemTypeString(CBE::item_t itemType) {
  static const struct ItemTypeValue {
    CBE::item_t itemType;
    const char* str;
  } ItemTypeValues[] {
      { CBE::ItemType::Unapplicable , "Unapplicable"  },
      { CBE::ItemType::Unknown      , "Unknown"       },
      { CBE::ItemType::Object       , "Object"        },
      { CBE::ItemType::Container    , "Container"     },
      { CBE::ItemType::Tag          , "Tag"           },
      { CBE::ItemType::Group        , "Group"         }
};
  const auto it = std::find_if(std::begin(ItemTypeValues) /* first */,
                               std::end(ItemTypeValues)   /* last */,
                               [itemType](const ItemTypeValue& ItemTypeValue) {
                                 return ItemTypeValue.itemType == itemType;
                               } /* p */);
  return (it != std::end(ItemTypeValues))? it->str : "unknown-item-type";
}

void Logic::printItem(const CBE::Item& item, bool printParentId) {
  std::cout << std::left << std::setw(31) << item.name()
            << std::left << std::setw(10) << itemTypeString(item.type())
            << std::left << std::setw(3)  << "id:"
            << std::left << std::setw(16) << item.id();
  if (printParentId) {
    std::cout << std::left << std::setw(2)  << "p:" << item.parentId();
  }
  std::cout << std::endl;
}

std::string Logic::containerName(CBE::ContainerPtr  container,
                                 bool               temporary) {
  std::ostringstream oss;
  oss << container->name() <<  " (" << (temporary? "(" : "")
      << container->id() << (temporary? ")" : "") << ")";
  return oss.str();
}

std::string Logic::objectName(CBE::ObjectPtr object,
                              bool           temporary) {
  std::ostringstream oss;
  oss << object->name() <<  " (" << (temporary? "(" : "")
      << object->id() << (temporary? ")" : "") << ")";
```

```
  return oss.str();
}
```

By adding this code your IDE should give you a warning that **AccountEventProtocol** is not known / defined. AccountEventProtocol is an implementation of the virtual class AccountEventProtocol in the includes->protocols folder in the includes from the library. The AccountEventProtocol is a kind of template class that we implement what we need from to be able to implement the callback delegate **CBE::AccountDelegatePtr** which is the smart pointer containing the callback after invoking an API request. To logIn there is a call on the CloudBackend class seen in includes->CloudBackend.h. This returns a **CloudBackendPtr** (smart pointer to an CloudBackend object), which is also the pointer we can see in AccountEventProtocol, callback virtual void onLogIn(…). To be able to call logIn we need an AccountDelegatePtr that takes an implementation of the template class AccountEventProtocol defined in includes->protocols.

Create file **AccountEventProtocol.h** (in your own implementation directory) and add:

```
#ifndef ACCOUNT_EVENT_PROTOCOL_H_
#define ACCOUNT_EVENT_PROTOCOL_H_

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

class AccountEventProtocol : public CBE::AccountEventProtocol {
 public:

  void onLogin(uint32_t atState, CBE::CloudBackendPtr cloudbackend) final;

  /** Gets called when the account status has changed (required). */
  void onError(CBE::persistence_t failedAtState, uint32_t code, std::string reason, std::string message)
final;

  AccountEventProtocol(Logic* ptr);

  Logic* cbeTL{};

  AccountEventProtocol(const AccountEventProtocol&) = delete;
  AccountEventProtocol& operator=(const AccountEventProtocol&) = delete;
};
#endif
```

Create file **AccountEventProtocol.cpp** and add:

```
#include "AccountEventProtocol.h"

void AccountEventProtocol::onLogin(uint32_t atState, CBE::CloudBackendPtr cloudbackend) {
  std::cout << "Account Login complete"
            << " - name: "<< cloudbackend->account()->username()
            << " - id: "<< cloudbackend->account()->userId() << std::endl;
  cbeTL->rootContainer = cloudbackend->account()->rootContainer();
  cbeTL->logic();
}

void AccountEventProtocol::onError(CBE::persistence_t failedAtState, uint32_t code, std::string reason,
std::string message) {
  std::cout << "Account Event Error: "
            << "Login was " << reason
            << " due to " << message << std::endl;
  cbeTL->programFinished();
}

AccountEventProtocol::AccountEventProtocol(Logic* ptr) {
  cbeTL = ptr;
}
```

onError has been defined to give you an error message in case the program is not able to access your account on the backend service.

As a last step you are now ready to compile Exercise 1 and run the generated program.

```
sh compile.sh 1       # compile exercise 1
sh run.sh 1           # run your compiled exercise 1
```

Exercise 2

In this exercise you will add a container to your account that we will later use to create an object in and then query this container. The name of this container is of course arbitrary and you can use whatever naming convention you like. Like with the AccountEventProtocol we will implement the **ItemEventProtocol** to be able to call the request and get a callback from the server when it is done. ItemEventProtocol and its corresponding delegate has the widest functionality span in the SDK so far. You can use an itemDelegate to listen to your account for incoming updates on the account, but that we will save for later.

You will add to the following source code files in the project folder:

- ItemEventProtocol.h
- ItemEventProtocol.cpp
- Logic.h
- Logic2.cpp

Create a file called **ItemEventProtocol.h** and copy paste this in to it:

```
#ifndef ITEM_EVENT_PROTOCOL_H_
#define ITEM_EVENT_PROTOCOL_H_

#include "CBE/Protocols/ItemEventProtocol.h"
#include "Logic.h"

class ItemEventProtocol : public CBE::ItemEventProtocol {
  public:
    /** Pointer to the Logic class. */
    Logic* cbeTL{};

    /** Returns the container with the id that have been set on the backend. */
    void onContainerAdded(CBE::ContainerPtr container) final;

    /** Gets called when an error occurred.*/
    void onItemError(CBE::ItemPtr container,
                                CBE::item_t type,
                     CBE::persistence_t operation,
                     CBE::persistence_t failedAtState,
                     uint32_t code,
                     std::string reason,
                     std::string message) final;

  /** Sets a pointer to the Logic class so that we can continue with API calls to the server. */
  ItemEventProtocol(Logic* ptr);

  /** Gets called when an Object has been added. */
  void onObjectAdded(CBE::ObjectPtr object) final;

  /** Gets called with the result of a query that has been done.
   * The result contains both the filter used for the query and the list with items as the result. */
  void onQueryLoaded(CBE::QueryResultPtr dir) final;

  /** Gets called if the server returns an error for the query. */
  void onLoadError(CBE::Filter filter,
                             uint32_t operation,
                   uint32_t code,
```

```
                    std::string reason,
                    std::string message) final;

    ItemEventProtocol(const ItemEventProtocol&) = delete;
    ItemEventProtocol& operator=(const ItemEventProtocol&) = delete;};
#endif
```

Next create file **ItemEventProtocol.cpp** and add the following:

```
#include "ItemEventProtocol.h"

#include <stdio.h>
#include <iostream>

/** Gets called when a Container has been added. */
void ItemEventProtocol::onContainerAdded(CBE::ContainerPtr container) {
  std::cout << "Container Added: " << container->name() << std::endl;
  cbeTL->logic();
}

/** Gets called when an error occurred. */
void ItemEventProtocol::onItemError(CBE::ItemPtr container, CBE::item_t type, CBE::persistence_t
operation, CBE::persistence_t failedAtState, uint32_t code, std::string reason, std::string message) {
  std::cout << "Item Event Error on: " << container->name() << std::endl;
  cbeTL->programFinished();
}

/** Gets called when an Object has been added. */
void ItemEventProtocol::onObjectAdded(CBE::ObjectPtr object) {
  std::cout << "Object Added: " << object->name() << std::endl;
  cbeTL->logic();
}

/** Gets called with a query result.*/
void ItemEventProtocol::onQueryLoaded(CBE::QueryResultPtr q) {
  std::cout << "Query loaded" << std::endl;
  cbeTL->continueProgram(q);
}

/** Gets called when loading a query fails. */
void ItemEventProtocol::onLoadError(CBE::Filter filter, uint32_t operation, uint32_t code, std::string
reason, std::string message) {
  std::cout << "Item Event Load Error: "
            << " was " << reason
            << " due to " << message << std::endl;


  cbeTL->programFinished();
}

ItemEventProtocol::ItemEventProtocol(Logic* ptr) {
  cbeTL = ptr;
}
```

The request we want to implement is a create on container.h so we are going to add a helper function in the Logic class.

Reuse the Logic.h and Logic1.cpp files from the previous task.

In **Logic.h** add the following lines under private:

```
// Exercise 2
void loadContainerContents(CBE::ContainerPtr container);
void printContainerContents(CBE::QueryResultPtr q);
CBE::ContainerPtr createContainer(CBE::ContainerPtr container);
CBE::ContainerPtr container{};
```

Copy the file Logic1.cpp in to the same folder and name the new file **Logic2.cpp**.

In **Logic2.cpp** add the following lines:

```
#include "ItemEventProtocol.h"


/* Exercise 2 */
CBE::ContainerPtr Logic::createContainer(CBE::ContainerPtr container) {
  std::cout << "Logic: Create container" << std::endl;
  const auto name = inquireString("Set name for Container");
  CBE::ItemDelegatePtr itemDelegate = std::make_shared(this);
  return container->create(name, itemDelegate);
}


void Logic::loadContainerContents(CBE::ContainerPtr container) {
  std::cout << "Getting sub-container " << containerName(container) << std::endl;
  CBE::ItemDelegatePtr itemDelegate = std::make_shared(this);
  CBE::Filter filter1;
  filter1.setDataType(CBE::ItemType::Container);
  container->query(filter1, itemDelegate);
}


void Logic::printContainerContents(CBE::QueryResultPtr q) {
  for (const auto& itemPtr : q->getItemsSnapshot()) {
    printItem(*itemPtr);
  }
}
```

Next step is to add to the switch / case function in logic(). From start() you will get access to the rootContainer in which we will create this container. Try implementing this yourself, otherwise replace logic() with the following example code inside the switch statement:

```
/* after login; load the top container list of items in cache */
case 1: {
  loadContainerContents(rootContainer);
  ++step;
  break;
}

/* Given the existing containers, should we create a new ? */
case 2: {
  printContainerContents(qResult);
  if (inquireBool("Do you want to create a new container", false)) {
    auto newContainer = createContainer(rootContainer);
    std::cout << "Container name: "
              << containerName(newContainer, true /* temporary */)
              << std::endl;
    container = newContainer;
  } else {
    continueToNextStep = true;
  }
  ++step;
  break;
}
```

When you have completed the file **Logic2.cpp** it is time to compile and run the code.

```
sh compile.sh 2       # compile exercise 2
sh run.sh 2           # run your compiled exercise 2
```

Exercise 3

In this exercise **query** and **createObject** will be implemented and used. You will add a lot of code for printing query results in different ways; one way for all the items in a container and one method for specifically the metadata added on an object. However, if you would like to combine these you are more then welcome to implement that. Apart from that, a getter for containers (from the recent query residing in the cache), query container and createObject helper functions will be added together with corresponding callbacks from the requests.

You will add the following source code files in the project folder:

- Logic.h
- Logic3.cpp

Next add the following in **Logic.h**:

```
// Exercise 3
  void printObjects(CBE::QueryResultPtr q);
  CBE::ContainerPtr selectContainer(const std::string& prompt);
  CBE::ObjectPtr createObject(CBE::ContainerPtr inContainer);
  CBE::ObjectPtr object{};
```

Next copy Logic2.cpp to **Logic3.cpp**

Add the following:

```
/* Exercise 3 */
void Logic::printObjects(CBE::QueryResultPtr q) {
  CBE::ObjectPtr tempObject{}; // nullptr;
  std::cout << "Printing Objects from query result: " << std::endl;
  for (const auto& itemPtr : q->getItemsSnapshot()) {
    if(itemPtr->type() == CBE::ItemType::Object) {
      printItem(*itemPtr);
      tempObject = cloudBackend->castObject(itemPtr);
      const auto keyValues = tempObject->keyValues();
      if(!keyValues.empty()) {
        for (const auto& keyValue : keyValues) {
          const auto& key      = keyValue.first;
          const auto& sdkValue = keyValue.second;
          const auto  indexed  = std::get<metadata_dataindex_indexed>(sdkValue);
          std::cout << "    " << key << " = " << "'"
                    << std::get<metadata_dataindex_value>(sdkValue)
                    << "'" << (indexed? " \t\t(indexed)" : "") << std::endl;
        }
      }  // if(!keyValues.empty())
    }   // if(itemPtr->type() == CBE::ItemType::Object)
  }
  if(!tempObject) {
    std::cout << "Sorry, no objects found in the Container!" << std::endl;
  }
}


CBE::ContainerPtr Logic::selectContainer(const std::string& prompt) {
  std::cout << "Select Container" << std::endl;
  const auto items = qResult->getItemsSnapshot();
  while (true) {
    const auto containerName =
      inquireString(prompt);
    for (const auto& item : items) {
      if(item->name() == containerName) {
        return cloudBackend->castContainer(item);
      }
    }
    std::cout << "Error: the container you asked for, \"" << containerName
              << "\", was not found; following are the options:" << std::endl;
    for(const auto& item : items) {
      printItem(*item);
    }
  }
}


CBE::ObjectPtr Logic::createObject(CBE::ContainerPtr inContainer) {
  std::cout << "Create Object" << std::endl;
  const int numOftags =
                      inquireInt("Set the number of Key/Value pairs you want");
  CBE::metadata_type keyValues;
  for(int i = 1; i <= numOftags; i++) {
```

```
    const auto tag = inquireString("Name of Key #" + std::to_string(i));
    const auto value =
            inquireString("Value for #" + std::to_string(i) + " '" + tag + "'");
    const auto indexed =
                      inquireBool("Make KeyValue pair #" + std::to_string(i) +
                                  " indexed or not (y indexed, n not indexed",
                                  true /* defaultVal */);
    keyValues[tag] = std::pair(value, indexed);
  }
  const std::string name = inquireString("Set name for Object");
  CBE::ItemDelegatePtr itemDelegate = std::make_shared(this);
  if (numOftags > 0) {
    return inContainer->createObject(name, itemDelegate, keyValues);
  } else {
    return inContainer->createObject(name, itemDelegate);
  }
}
```

Next step is to add to the switch / case function in logic(). From start() you will get access to the rootContainer in which we will create this container. Try implementing this yourself, otherwise add to logic() with the following example code:

```
/* Look in the target container */
case 3: {
  ++step;
  loadContainerContents(rootContainer);
  break;
}

/* print list of items; select target container; create object */
case 4: {
  printContainerContents(qResult);
  std::cout << "Item count: " << qResult->getItemsSnapshot().size()
            << std::endl;
  container =
      selectContainer(
        "In which Container do you want to create the new object");
  std::cout << "Container: " << containerName(container) << std::endl;
  object = createObject(container);
  std::cout << "Designated name: "
            << objectName(object, true /* temporary */) << std::endl;
  ++step;
  break;
}

/* object created; load the top container list of items in cache */
case 5: {
  std::cout << "Object name: " << objectName(object) << std::endl;
  loadContainerContents(container);
  ++step;
  break;
}

/* print list of objects and their key/value data */
case 6: {
  printObjects(qResult);
  ++step;
    logic();
    break;
}
```

Compile and run the code.

```
sh compile.sh 3       # compile exercise 3
sh run.sh 3           # run your compiled exercise 3
```

To reset your account, do implement on your own, a way to clean up containers and objects that have been created.

Scripts

Finally add the following two scripts **compile.sh** and **run.sh**. If you want to use your own scripts do so, but if you just want to copy paste use the following and read the comments in the script:

compile.sh

```
#!/bin/sh
# Copyright © CloudBackend AB 2022.
# compile.sh #
# version 2022-03-24
# This compiler script has sections corresponding to
# the different exercises in the Tutorial # on the website.
# Submit the number of the Exercise to compile.
# Example: sh compile.sh 1

if [ $# -gt 0 ]
then
    exe=$1
else
    exe="0"
fi
ARCH=`uname -m`
echo "computer architechture:" ${ARCH}
case "$ARCH" in
    "x86_64")
    COMPILER_COMMAND="g++ -std=c++17 -pthread "
    # your path to the copy of the SDK binary lib
    # libCBE="../../SDK/Linux_x86/libcb_sdk.so"    # export LD_LIBRARY_PATH="../../SDK/Linux_x86/"
    libCBE="../../SDK/Linux_x86/libcb_sdk.a"
    WARNINGS="-Wpedantic -Wall -Wextra -Weffc++ -Wsuggest-override -Wno-unused-parameter"
    ;;

    *)
    echo ${ARCH} "platform is not supported in this release"
    exit 1
    ;;
esac

echo "compile exercise:" ${exe}
case "${exe}" in
    1)
    # Tutorial exercise 1:
    ${COMPILER_COMMAND} ${WARNINGS} -o tut${exe} Main.cpp Logic${exe}.cpp AccountEventProtocol.cpp
${libCBE} -I "../../include"
    command_result=$?
    ;;

    2|3)
    # Tutorial exercise:
    ${COMPILER_COMMAND} ${WARNINGS} -o tut${exe} Main.cpp Logic${exe}.cpp AccountEventProtocol.cpp
ItemEventProtocol.cpp ${libCBE} -I "../../include"
    command_result=$?
    ;;
```

```
    *)
    # Default.
    echo "syntax : "$0" <1-3> "
    echo "example: "$0" 1"
    exit 2
    ;;
esac

if [ ${command_result} -eq 0 ]
then
    echo "to run use: sh run.sh" ${exe}
else
    echo "compilation had errors"
fi
```

* note that two actions are needed in this script, first the ${libCBE} variable needs to point to where you have the binary, second is the -I flag which needs to point to where you have the include folder for the SDK. If you use the libcb_sdk.so, prior to running the compiled code, set export LD_LIBRARY_PATH="../../SDK/Linux_x86/"

run.sh

```
#!/bin/sh
# run.sh #
# This run script has sections corresponding to the different exercises in the Tutorial on the website.
# Submit the number of the exercise to compile.
# example: sh run.sh 1

echo "CloudBackend SDK is provided under a limited evaluation licence "
echo "that is not for production use."

if [ $# -gt 0 ]
then
  exe=$1
else
  exe="0"
fi
ARCH=`uname -m`
echo "computer architechture:" ${ARCH}
case "$exe" in
  1|2|3)
  echo "run exercise:" ${exe}
  ./tut${exe}
  ;;

  *)
  # Default.
  echo
  echo "syntax: "$0" <1-3> "
  echo "example: sh run.sh 1"
  exit 1
  ;;
esac
```

Acknowledgement

Many thanks to Forware AB for valuable contribution to the example code in this Tutorial.

Downloads

The resulting source code files can also be downloaded from the Tutorial repository.