COSBench Adaptor Development Guide

Version 2.5.3

April, 2013

Wang, Yaguang

This document describes how to extend COSBench to support new storage systems by developing corresponding adaptors. It depicts the model assumed by COSBench, lists interfaces to be implemented in adaptors, and demonstrates how to adopt new adaptors into the system.

Document Number: 328792-001US

0413/RJM/MESH/PDF   328792-001US

Contents

Revision History

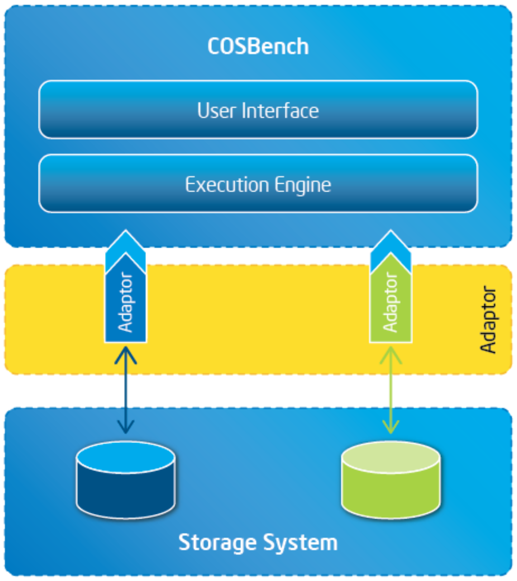
Revision	Date	Description
2.0	November 9, 2012	Initial version for v2.0
2.1	November 22, 2012	Add diagrams and samples
2.2	February 19, 2013	Add deployment section
2.3	March 18, 2013	Add section 5.1.10 for “abort” method in storageAPI
2.4	March 28, 2013	• Add call flow for authAPI and storageAPI interfaces • Add explanation for arguments in interface definition • Reword some sentences
2.5	April 10, 2013	• Change folder name from “modules” to “plugins”
2.5.3	August 09, 2013	• Add section 10.1 to explain how to generate adaptor bundle. • Miscellaneous corrections.

1 Introduction

COSBench is an open-source distributed benchmark tool used to quantify the performance of cloud object storage systems. That data enables users to compare various cloud object stores, evaluate hardware and software stacks, and identify bottlenecks for performance tuning.

Users can get more information about COSBench, download the installation package and other materials, and contribute at https://github.com/intel-cloud/cosbench. For information about installation, configuration, and use of the tool, see the “COSBench User Guide.”

COSBench natively supports OpenStack* Swift and Amplidata AmpliStor* v2.3, v2.5, and 3.1. COSBench is based on a modular, extensible design that includes pre-defined APIs for the development of adaptors to support additional storage systems.



In this document, code samples use the character string “xxx” to represent the version of COSBench being used. In practice, users would substitute the actual COSBench version in place of “xxx” (e.g., 0.3.0).

2 Glossary

Adaptor: a bundle that implements the Auth API and/or Storage API

API: the interface exposed from COSBench to enable extensibility, currently including the Auth API and Storage API

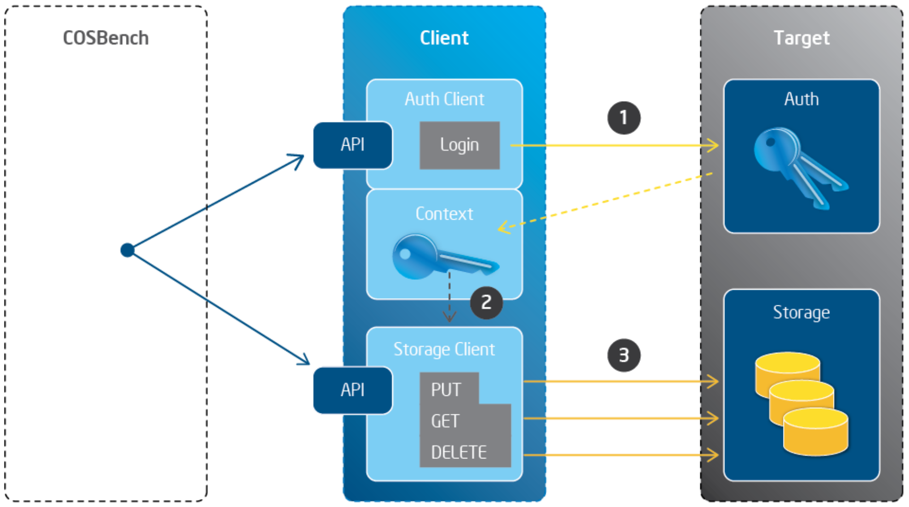
Bundle (also referred to as OSGi Bundle or plug-in): a JAR file compliant with the Open Services Gateway initiative (OSGi) specification

3 General Concepts

3.1 Overview

The connection of COSBench to a storage system and its basic operation are depicted below; there are three major components:

- API
- Client
- Context



1. COSBench calls Login() through an authentication client using credentials, which are passed to the target authentication server to authenticate the credentials passed in. If authentication succeeds, a corresponding token is returned and stored in authentication context.
2. COSBench calls PUT/GET/DELETE() through the storage client, and the storage client acquires the authentication token from context.
3. After obtaining the token, the storage client makes calls to the target storage server.

3.2 APIs

COSBench defines two sets of APIs to adapt new storage systems:

- Auth API, to handle authentication-related work
- Storage API, to handle data-access-related work

The implementation of both APIs can be included in one OSGi bundle or separated into two OSGi bundles. If the authentication mechanism is a general-purpose one such as Open Auth or Kerberos, best practices call for implementing the Auth API in an individual bundle.

3.3 Client

Client is a class that encapsulates all interactions with target storage or authentication systems.

3.4 Context

Normally, it’s necessary to exchange information between an authentication client and a storage client, such as an authenticated token returned from the authentication system; that token is required by the storage client for subsequent storage operations. That type of shared information is maintained in one context, as a key-value pair.

4 Auth API

4.1 Credential

Credential consists of a parameter list that includes all information related for authentication on a storage client (e.g., username, password, and authentication URL). Different storage systems may use different parameter lists.

4.2 Interfaces

```
public void init(Config config, Logger logger);
public void dispose();
public Context getParms();
public AuthContext login();
```

4.2.1 init()

```
    /**
     * Initializes an <code>Auth-API</code> with parameters contained in the
     * given <code>config</code>, whose content depends on the specific Auth
     * type. Normally, it will also initialize one client for authentication.
     *
     * @param config
     *            - one instance from com.intel.cosbench.config.Config, which
     *            includes parameters for authentication, and it will be passed
     *            from execution engine.
     * @param logger
     *            - one instance from com.intel.cosbench.log.Logger, which
     *            delivers logging capabilities to Auth-API, and it will be passed
     *            from execution engine.
     */
public void init(Config config, Logger logger);
```

4.2.2 dispose()

```
/**
 * release the resources held by the Auth API.
 */
public void dispose();
```

4.2.3 getParms()

```
/**
 * retrieve parameters and current settings used by the AuthAPI.
 * @return Context - one com.intel.cosbench.context.Context instance which contains all
parameters configured for the authentication mechanism.
 */
public Context getParms();
```

4.2.4 login()

```
/**
 * trigger backend authentication mechanism.
 * @return AuthContext - one com.intel.cosbench.context.AuthContext instance which contains all
parameters configured for the authentication mechanism if authentication is successful, and
otherwise, an exception will be raised.
 */
public AuthContext login();
```

5 Storage API

5.1 Interfaces

```
public void init(Config config, Logger logger);
public void dispose();
public Context getParms();
public void setAuthContext(AuthContext info);
public InputStream getObject(String container, String object, Config config);
public void createContainer(String container, Config config);
public void createObject(String container, String object, InputStream data, long length, Config
config);
public void deleteContainer(String container, Config config);
public void deleteObject(String container, String object, Config config);
```

5.1.1 init()

```
    /**
     * Initializes a <code>Storage-API</code> with parameters contained in the
     * given <code>config</code>, whose content depends on the specific storage
     * type. Normally, it will also initialize one client for storage access.
     *
     * @param config
     *           - one instance from com.intel.cosbench.config.Config, which
     *           includes parameters for authentication, and it will be passed
     *           from execution engine.
     * @param logger
     *           - one instance from com.intel.cosbench.log.Logger, which
     *           delivers logging capabilities to Storage-API, and it will be passed
     *           from execution engine.
     */
    public void init(Config config, Logger logger);
```

5.1.2 dispose()

```
/**
 * release the resources held by the Storage API.
 */
public void dispose();
```

5.1.3 getParms()

```
/**
 * retrieve parameters and current settings used by the StorageAPI.
 * @return Context - one Context instance which contains all parameters configured for the
storage.
 */
public Context getParms();
```

5.1.4 setAuthContext()

```
/**
 * associate authenticated context with Storage API for further storage operations.
 * @param info - one AuthContext instance, normally, it's the return from login() in Auth API.
 */
public void setAuthContext(AuthContext info);
```

5.1.5 getObject()

```
/**
 * download an object from a container.
 *
 * @param container - the name of a container.
 * @param object - the name of an object to be downloaded.
 * @param config - the configuration used for this operation.
 * @return inputStream - the inputStream of the object content. If null that means the object
doesn't
 * exist or something bad happened.
 */
public InputStream getObject(String container, String object, Config config);
```

5.1.6 createContainer()

```
/**
 * create a container.
 *
 * @param container - the name of a container.
 * @param config - the configuration used for this operation.
 */
public void createContainer(String container, Config config);
```

5.1.7 createObject()

```
/**
 * upload an object into a container.
 *
 * @param container - the name of a container.
 * @param object - the name of an object to be uploaded.
 * @param data - the inputStream of the object content.
 * @param length - the length of object content.
 * @param config - the configuration used for this operation.
 */
public void createObject(String container, String object, InputStream data, long length, Config
config);
```

5.1.8 deleteContainer()

```
/**
 * delete a container.
 *
 * @param container - the name of a container to be deleted.
 * @param config - the configuration used for this operation.
 */
public void deleteContainer(String container, Config config);
```

5.1.9 deleteObject()

```
/**
 * delete an object.
 *
 * @param container - the name of a container.
 * @param object - the name of an object to be deleted.
 * @param config - the configuration used for this operation.
 */
public void deleteObject(String container, String object, Config config);
```

5.1.10 abort()

```
/**
  * Aborts the execution of an on-going storage operation (HTTP request) if
  * there is one.  The method expects to provide  one approach to abort outstanding
  * operation gracefully when the worker hits some termination criteria.
  */
public void abort();
```

6 Adaptor Development

6.1 Overview

An adaptor is actually an OSGi* (http://www.osgi.org) bundle. A few IDEs support the development of OSGi bundles, and the IDE used for COSBench is Eclipse* SDK 3.7 Indigo (http://www.eclipse.org).

Developing a new adaptor involves following steps:

1. Import a few dependencies.
2. Set up source code structure.
3. Modify configurations.

6.2 Plug-in Dependencies

6.2.1 Third-party Libraries

```
        - other libraries needed based on abcStor specific requirements.
        - com.springsource.org.apache.commons.codec-1.30.jar
        - org.apache.httpcomponents.httpclient_4.1.3.jar
        - org.apache.httpcomponents.httpcore_4.1.4.jar
        - cosbench-api_xxx.jar
        - cosbench-config_xxx.jar
        - cosbench-http_xxx.jar
        - cosbench-log_xxx.jar
```

6.2.2 COSBench Bundles

Two primary bundling approaches are supported, corresponding to the two APIs to be implemented. These approaches are illustrated below using the sample bundles abcStor and abcAuth. Other approaches, such as leveraging existing auth or storage bundles, are not included.

- Two in one:
  - abcStor+abcAuth bundle
- One plus one:
  - abcStor bundle (Storage API)

- abcAuth bundle (AuthAPI)

6.3 Source Code Structure

Source code typically makes use of two folders:

- The "*api*" folder includes all required API declarations, as well as calls to functionalities included in the "client" folder.
- The "client" folder includes storage/auth-specific implementation. Classes in this folder don't need any COSBench bundles, based on considerations for the following two cases:
    - If there is tested code in an adaptor developer site, the developer can put the code into the "client" folder and add the necessary wrapper in "api" to call into it.
    - For a fresh, new implementation, the developer can focus on storage-specific implementations in client and build tests based on the contents of this folder, without the involvement of COSBench code or OSGi plug-in-related configurations, simplifying development efforts.

The reference code structure given below uses abcStor as an example; the full abcStor project is stored in the "*plugin"* folder in the COSBench package.

```
| src
  | com.abc
       | api
            | abcStorage.java
            | abcStorFactory.java
       | client
          | abcStorClient.java
          | abcStorConstants.java
          | abcStorClientException.java
```

6.4 Configurations

6.4.1 META-INF\MANIFEST

Below is one sample for the MANIFEST file; adaptor developers must modify this code according to their specific development requirements. Normally, the adaptor developer must review the "Import-Package" list.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Abc Storage Client Bundle
Bundle-SymbolicName: cosbench-abcstor
Bundle-Version: 0.3.1.0
Bundle-Vendor: Abc Co.
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

```
Import-Package: com.intel.cosbench.api.auth,
 com.intel.cosbench.api.context,
 com.intel.cosbench.api.storage,
 com.intel.cosbench.client.http,
 com.intel.cosbench.config,
 com.intel.cosbench.log,
 org.apache.commons.codec;version="[1.3.0,2.0.0)",
 org.apache.commons.codec.net;version="[1.3.0,2.0.0)",
 org.apache.http;version="[4.1.4,5.0.0)",
 org.apache.http.client;version="[4.1.3,5.0.0)",
 org.apache.http.client.methods;version="[4.1.3,5.0.0)",
 org.apache.http.conn;version="[4.1.3,5.0.0)",
 org.apache.http.entity;version="[4.1.4,5.0.0)",
 org.apache.http.message;version="[4.1.4,5.0.0)",
 org.apache.http.params;version="[4.1.4,5.0.0)",
 org.apache.http.util;version="[4.1.4,5.0.0)"
```

6.4.2 META-INF\spring\plugin-context.xml

The sample code below illustrates “one plus one” bundling mode; there are two plug-in projects.

abcAuth

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:osgi="http://www.springframework.org/schema/osgi"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/osgi
        http://www.springframework.org/schema/osgi/spring-osgi.xsd">

    <bean name="authFactory" class="com.abc.api.abcAuth.AbcAuthFactory" />

    <osgi:service ref="authFactory" context-class-loader="service-provider"
              interface="com.intel.cosbench.api.auth.AuthAPIFactory">
    </osgi:service>

</beans>
```

abcStor

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:osgi="http://www.springframework.org/schema/osgi"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/osgi
        http://www.springframework.org/schema/osgi/spring-osgi.xsd">

    <bean name="storageFactory" class="com.abc.api.abcStor.AbcStorageFactory" />

    <osgi:service ref="storageFactory" context-class-loader="service-provider"
              interface="com.intel.cosbench.api.storage.StorageAPIFactory">
    </osgi:service>

</beans>
```

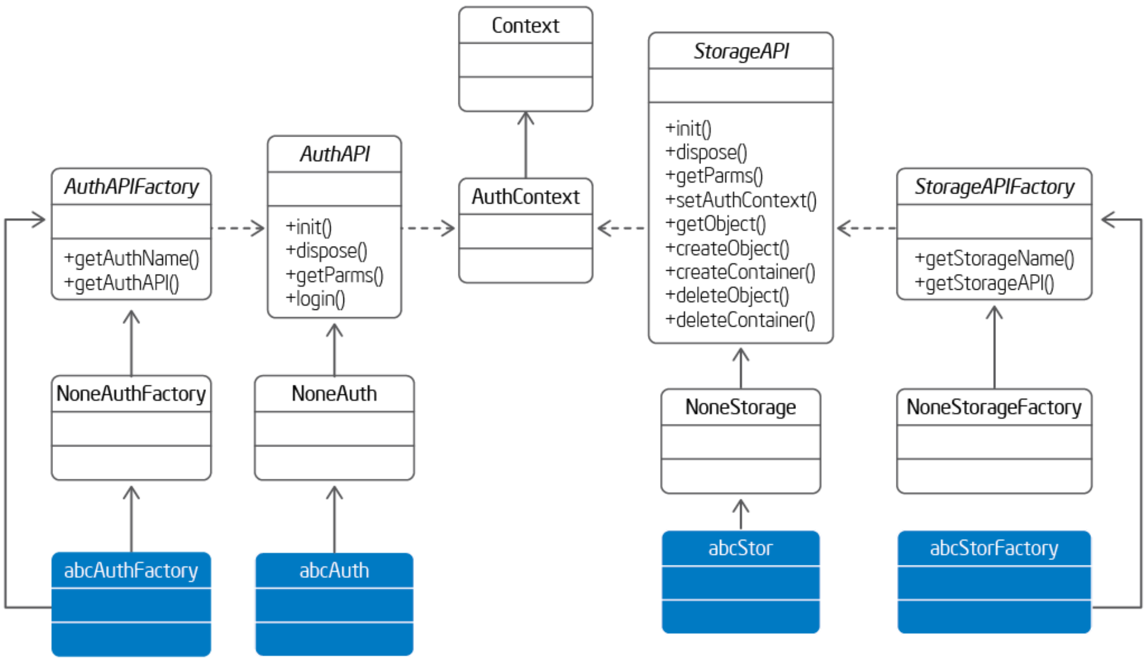
The code sample below is for “two in one” bundling mode; there is one plug-in project, and two OSGi services are defined in one plugin-context.xml:

abcStor + abcAuth

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:osgi="http://www.springframework.org/schema/osgi"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/osgi
        http://www.springframework.org/schema/osgi/spring-osgi.xsd">

    <bean name="authFactory" class="com.abc.api.abcAuth.AbcAuthFactory" />

    <osgi:service ref="authFactory" context-class-loader="service-provider"
              interface="com.intel.cosbench.api.auth.AuthAPIFactory">
    </osgi:service>

    <bean name="storageFactory" class="com.abc.api.abcStor.AbcStorageFactory" />

    <osgi:service ref="storageFactory" context-class-loader="service-provider"
              interface="com.intel.cosbench.api.storage.StorageAPIFactory">
    </osgi:service>

</beans>
```

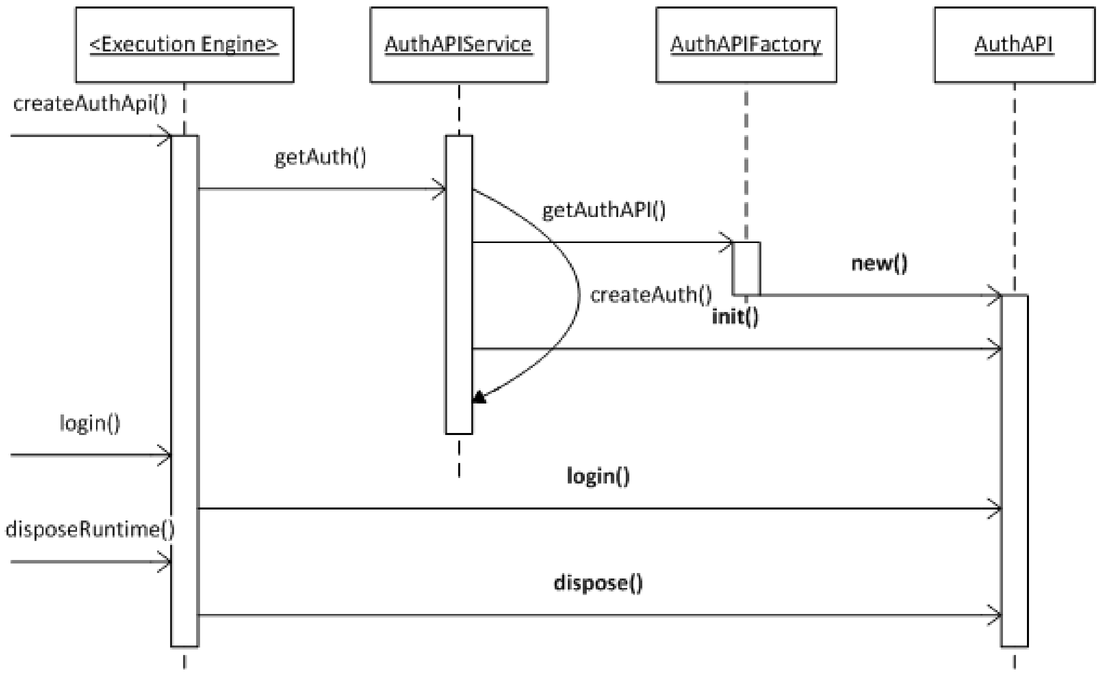
7 Class Diagram and Call Flow

The following class diagram shows the relationship between classes related to adaptor development.



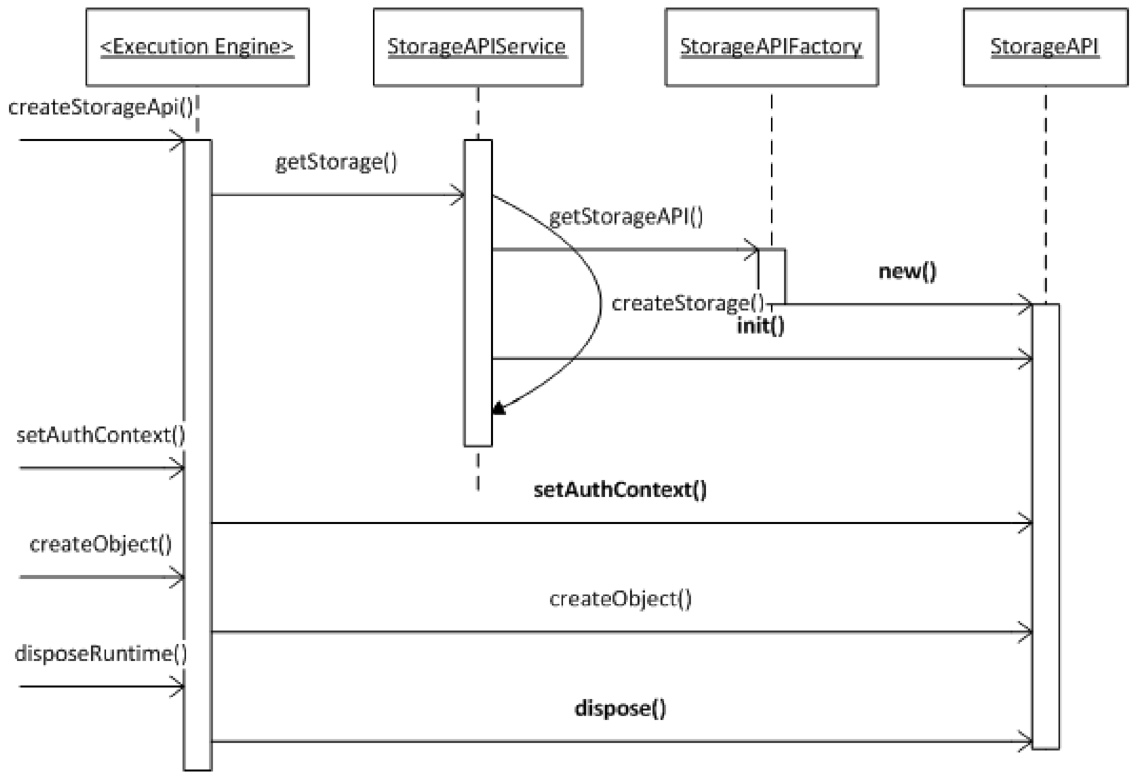
7.1 AuthAPI Lifecycle

The sequence for how authAPI (abcAuth) is initialized, performed, and disposed is illustrated below; the procedure is driven by the COSBench execution engine.



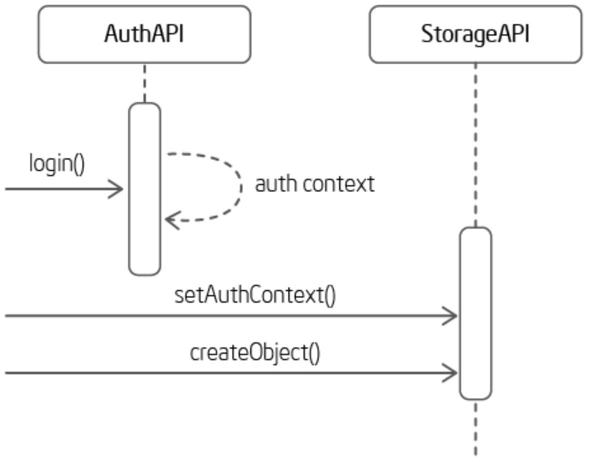
7.2 StorageAPI Lifecycle

The sequence for how storageAPI (abcStorage) is initialized, performed, and disposed is illustrated below; the procedure is driven by the COSBench execution engine.



7.3 AuthAPI and StorageAPI Interaction

Inside COSBench, the execution engine calls login() on AuthAPI (abcAuth) to authenticate the user; after successfully getting valid auth context, it calls setAuthContext() to pass the returned auth context to StorageAPI (abcStor). At this time, COSBench can issue object-storage-related operations through information from auth context. The flow is illustrated below.



8 Parameters

A new auth or storage adaptor could require different parameters, and those parameters would be defined in the "config" attribute within the workload configuration file as key-value pairs separated by semicolons, as shown below (using abcAuth as an example):

```
<auth type="abcauth"
config="username=test;password=testing;auth_url=http://192.168.10.1:8080/auth/v1.0;timeou
t=10000" />
```

Normally, the parameters will be read in "init()" to help initialize the client.

9 Sample Project

There is a sample project shipped with the release package within the "ext" folder; one auth sample and one storage sample are included.

```
libs
        com.springsource.org.apache.commons.codec-1.3.0.jar
        org.apache.httpcomponents.httpclient_4.1.3.jar
        org.apache.httpcomponents.httpcore_4.1.4.jar
plugins
        cosbench-api_xxx.jar
        cosbench-config_xxx.jar
        cosbench-http_xxx.jar
        cosbench-log_xxx.jar
```

```
| adaptor
    | abc-auth
        | src
            | com.abc
                | api
                    | abcAuth.java
                    | abcAuthFactory.java
                | client
                    | abcAuthClient.java
                    | abcAuthConstants.java
                    | abcAuthClientException.java
        | META-INF
    | abc-stor
        | src
            | com.abc
                | api
                    | abcStorage.java
                    | abcStorFactory.java
            | client
                | abcStorClient.java
                | abcStorConstants.java
                | abcStorClientException.java
    | META-INF
```

10 Deployment

A few additional steps are needed to deploy adaptor bundles into COSBench.

10.1 Generate Adaptor Bundle

In Eclipse IDE, to generate the plugins by right clicking the project, and selecting "export... -> Plug-in Development -> Deployable plugins and fragments", and setting the "Directory" to "dist\osgi" folder, then the plugins library will be generated at "dist\osgi\plugins" folder. E.g., for “abc-stor” bundle, one bundle named “cosbench-abcstor_xxx.jar” will be generated.

10.2 Register Adaptor Bundle

The OSGi configuration file is located at conf/.driver/config.ini; a new adaptor bundle must register itself in it. Normally, COSBench-related plug-ins are near the end of the file, shown in the following:

```
plugins/cosbench-castor,\
plugins/cosbench-log4j,\
plugins/cosbench-log@6\:start,\
plugins/cosbench-config@6\:start,\
plugins/cosbench-http@6\:start,\
plugins/cosbench-core@7\:start,\
plugins/cosbench-api@7\:start,\
plugins/cosbench-mock@7\:start,\
plugins/cosbench-ampli@7\:start,\
plugins/cosbench-swift@7\:start,\
plugins/cosbench-keystone@7\:start,\
plugins/cosbench-swauth@7\:start,\
plugins/cosbench-abcstor@7:\start,\
plugins/cosbench-driver@7\:start,\
plugins/cosbench-tomcat@7\:start,\
plugins/cosbench-core-web@7\:start,\
plugins/cosbench-driver-web@7\:start
```

10.3 Update Bundle List

The launch script for the driver—"**start-driver.sh**"—will check whether the bundle is loaded successfully. In order to ensure that the adaptor bundle is loaded successfully, the script should be modified by updating its bundle list as follows:

```
OSGI_BUNDLES="cosbench-log_${VERSION} cosbench-tomcat_${VERSION} cosbench-
config_${VERSION} cosbench-core_${VERSION} cosbench-core-web_${VERSION} cosbench-
api_${VERSION} cosbench-http_${VERSION} cosbench-mock_${VERSION} cosbench-
ampli_${VERSION} cosbench-swift_${VERSION} cosbench-abcstor_${VERSION} cosbench-
keystone_${VERSION} cosbench-driver_${VERSION} cosbench-driver-web_${VERSION}"
```