ONAP Application Controller (APPC) Client Library Guide

Revision	**Version 1.0.0**
Revision Date	**22 August 2017**

Revision History

Date	Revision	Author	Changes
2017-08-22	1.0.0	Paul Miller	First draft consistent for Amsterdam Release

Table of Contents

1. Introduction 4
1.1. Target Audience 4
1.2. Related Documentation 4

1.3. Command-line Conventions 5
1.4. Text Conventions 5
1.5. Authors and Contributors 6
1.6. Terms and Acronyms 6
2. Client Library Background 8
2.1. About the Client Library 8
2.2. Consumer Logic 8
2.3. VNF Lifecycle Management API 8
2.4. APP-C Client Library Flow 9
2.4.1. Asynchronous Flow 9
2.4.2. Synchronous Flow 9
3. Client Library Usage 11
3.1. Jar Files 11
3.2. Initialization 11
3.3. Shutdown 11
3.4. Invoking LCM Commands 12
4. Client API 13
4.1. LifeCycleManagerStateful Interface 13
4.2. API documentation 13
4.3. appc-provider-lcm 14
4.4. Methods 14

1. Introduction

1.1. Target Audience

This document is for an advanced technical audience, which includes engineers and technicians. Document revisions occur with the release of new software versions.

1.2. Related Documentation

For additional information, see the ONAP Application Controller (APPC) API Guide.

The following sections describe the conventions this document uses, including notices, text conventions, and command-line conventions.

1.3. Command-line Conventions

The following table lists possible elements in a command-line path.

Convention	Description
Brackets [ ]	This is used for optional items.
Braces { }	This indicates choices separated by pipe (|) for sets from which only one is selected. For example: {even|odd}
Blue text	This indicates a link in this document online.

1.4. Text Conventions

The following table lists text conventions in this document.

Convention	Description

Monospace font with blue shading	This font indicates sample codes, screenshots, or elements. For example:
	`contact": {`
	`        "contactType": "USER",`
	`        "source": "app1",`
	`}`

Italics	Emphasizes a point or denotes new terms defined in the text. Indicates an external book title reference.
Numeric	A number composed of digits 0 through 9.
Text	Any combination of alphanumeric characters. New items in RED

1.5. Authors and Contributors

The following table lists the persons who are authors and contributors to this document.

Contributors	
Borislav Glozman	Margrethe Fossberg
Paul Mellor	John Buja

1.6. Terms and Acronyms

The following table defines terms and acronyms used in this document.

Term or Acronym	Definition
AAI	Active and Available Inventory
AAF	Authentication & Authorization Framework
AJSC	AT&T Java Service Container
API	Application Programming Interface
APPC	Application Controller
SDC	Service Design and Creation
DCAE	Data Collection Analytics and Events
DG	Directed Graph
DNS	Domain Name System
EELF	Event and Error Logging Framework
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
IAAS	Infrastructure As A Service
I/O	Input/Output
JMS	Java Messaging Service
JSON	JavaScript Object Notation
LAN	Local Area Network
LRM	Local Resource Monitor
SO	Service Orchestrator
NOD	Network on Demand
ODL	OpenDaylight
ONAP	Open Network Application Platform
OS	Operating System
PO	Platform Orchestrator
RCT	Reference Connection Tool
RO	Resource Orchestrator

Term or Acronym	Definition
SDN-C	Software Defined Network - Controller
SDN-GP	Software Defined Network – Global Platform
SME	Subject Matter Expert
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol
SOT	Source Of Truth (ext. system where data object originates)
SSH	Secure Shell
TCP	Transmission Control Protocol
TPS	Transactions per Second
UEB	Universal Event Broker
vCE	virtual CE (Customer Edge) router
vPE	virtual PE (Provider Edge) router
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
vSCP	Virtualized Service Control Point
WAN	Wide Area Network
WUI	Web User Interface
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

2. Client Library Background

This guide discusses the Application Controller (APPC) Client Library and how to use it.

2.1. About the Client Library

The APPC client library provides consumers of APPC capabilities with a strongly-typed Java interface and encapsulates the actual interaction with the APPC component over an asynchronous messaging channel such as UEB.

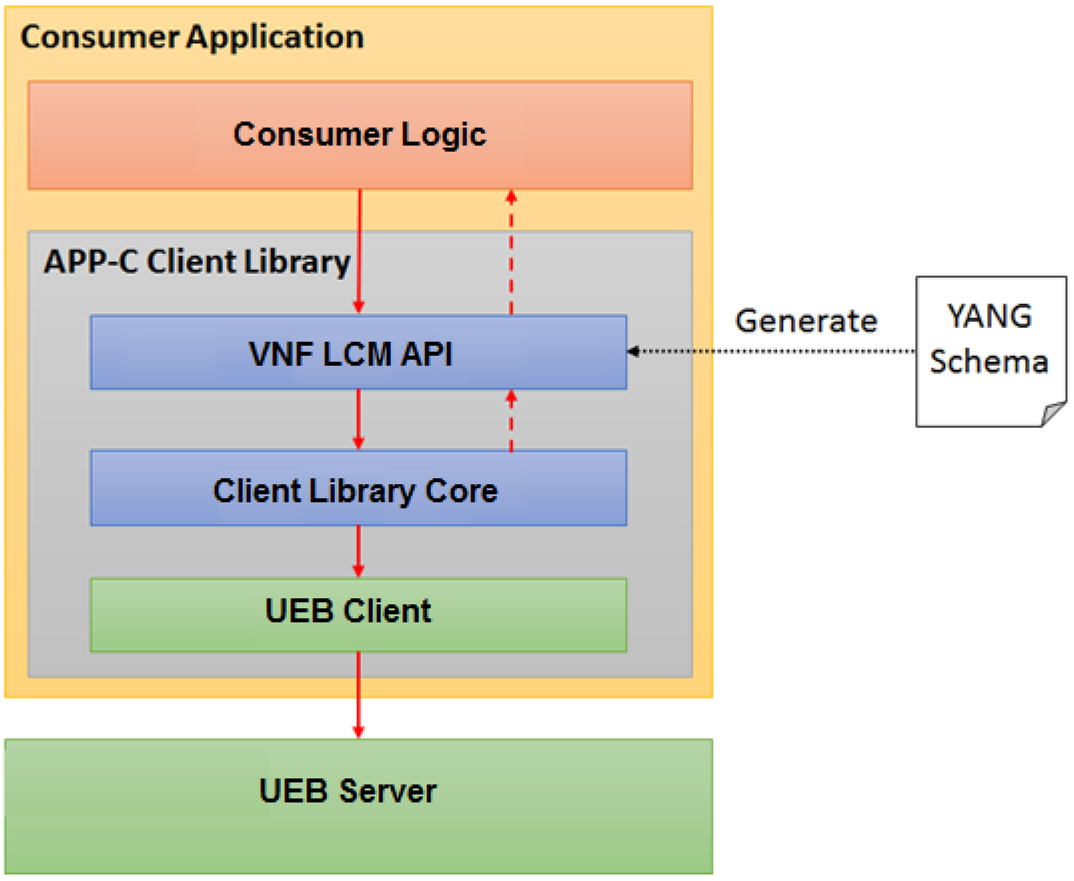
2.2. Consumer Logic

The client application that consumes APPC's capability for VNF lifecycle management (the APPC client library) can be implemented against the lightweight and strongly-typed Java API exposed by the APPC client library. The library does not try to impose architectural constraints upon clients, but instead provides support for different options and styles of API. It is the responsibility of the client application to select the most suitable paradigm to use; for example, a client may choose to use blocking calls as opposed to asynchronous notifications.

2.3. VNF Lifecycle Management API

The API represents a relatively thin layer that consists mainly of business interfaces with strongly-typed APIs and a data object model created for the convenience of the consumer application.
The original YANG schema used by the APPC component and the underlying MD-SAL layer on the server-side generates these artifacts.

2.4. APP-C Client Library Flow



2.4.1. Asynchronous Flow

- The APPC Client Library is called using an asynchronous API using a full command object, which is mapped to a JSON representation.
- The APPC client calls the UEB client and sends the JSON command to a configured topic.
- The APPC client pulls response messages from the configured topic.
- On receiving the response for the command, the APPC client runs the relevant callback method of the consumer ResponseHandler.

2.4.2. Synchronous Flow

- The APPC Client Library is called using a synchronous API using a full command object, which is mapped to a JSON representation.
- The APPC client calls the UEB client and sends the JSON command to a configured topic.
- The APPC client pulls response messages from the configured topic.

- On receiving the *final* response for the command, the APPC client returns the response object with a final status.

3. Client Library Usage

3.1. Jar Files

The Java application that runs the APPC client kit uses the following jar files:

- com.att.appc.client.client-kit
- com.att.appc.client.client-lib

The client library JAR files are located in the repository under
`//gerrit.onap.org/r/p/appc.git/appc-client/`

3.2. Initialization

Initialize the client by calling the following method:

```
AppcClientServiceFactoryProvider.getFactory(AppcLifeCycleManagerServiceFactory.class).createL
ifeCycleManagerStateful()
```

Specify the following configuration properties as method parameters:

- "topic.read"
- "topic.read.timeout"
- "topic.write"
- "client.key"
- "client.secret"
- "client.name"
- "client.name.id"
- "poolMembers"
- “client.response.timeout”
- “client.graceful.shutdown.timeout”

3.3. Shutdown

Shutdown the client by calling the following method:

```
void shutdownLifeCycleManager(boolean isForceShutdown)
```

If the `isForceShutdown` flag is set to false, the client shuts down as soon as all responses for pending requests are received, or upon configurable timeout.
(`client.graceful.shutdown.timeout`).
If the `isForceShutdown` flag is set to true, the client shuts down immediately.

3.4. Invoking LCM Commands

Invoke the LCM commands by:

- Creating input objects, such as AuditInput, LiveUpgradeInput, with relevant command information.
- Executing commands asynchronously, for example:

```
void liveUpgrade(LiveUpgradeInput liveUpgradeInput, ResponseHandler<LiveUpgradeOutput>
listener) throws AppcClientException;)
```

In this case, client should implement the ResponseHandler<T> interface.

- Executing commands synchronously, for example:

```
LiveUpgradeOutput liveUpgrade(LiveUpgradeInput liveUpgradeInput) throws
AppcClientException;)
```

4. Client API

After initializing the client, a returned Object of type LifeCycleManagerStateful defines all the Life Cycle Management APIs supported by APPC.

The interface contains two definitions for each RPC: one for Asynchronous call mode, and one for Synchronous.

In Asynchronous mode, client consumer should provide a callback function of type:

```
        ResponseHandler<RPC-NAMEOutput>
```

where `RPC-NAME` is the command name, such as Audit or Snapshot.

There may be multiple calls to the ResponseHandler for each response returned by APPC. For example, first 100 ‘accept’ is returned, then 400 ‘success’.

4.1. LifeCycleManagerStateful Interface

Generated from the APPC Yang model, this interface defines the services and request/response requirements for the ECOMP APPC component. For example, for LCM Command Audit, the following is defined:

```
@RPC(name="audit", outputType=AuditOutput.class)

AuditOutput audit(AuditInput auditInput) throws AppcClientException;
```

For a Synchronous call to Audit, the consumer thread is blocked until a response is received or a timeout exception is thrown.

```
@RPC(name="audit", outputType=AuditOutput.class)

void audit(AuditInput auditInput, ResponseHandler<AuditOutput> listener) throws
AppcClientException;
```

For an Asynchronous call to Audit, a callback should be provided so that when a response is received the listener is called.

4.2. API documentation

The API documentation is also available as a swagger page generated from files at /client-kit/target/resources.

4.3. appc-provider-lcm

This defines the services and request/response requirements for the APPC component.

4.4. Methods

The methods should match the actions described in the LCM API Guide.   For each method:

Consumes

This API call consumes the following media types using the **Content-Type** request header:

- `application/json`

Request body

The request body is the action name followed by Input (e.g., AuditInput)

Return type

The return type is the action name followed by Output (e.g., OutputInput)

Produces

This API call produces the following media types according to the **Accept** request header; the **Content-Type** response header conveys the media type.

- `application/json`

Responses

200	Successful operation
401	Unauthorized
500	Internal server error