
HP Customer Care Dashboard

Version 1.0



MbbQoe Customization Configuration Guide

Edition: 1.1

May 2014

© Copyright 2014 Hewlett-Packard Development Company, L.P.

Legal Notices

Warranty

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

License Requirement and U.S. Government Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

© Copyright 2014 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe®, Acrobat® and PostScript® are trademarks of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Java™ is a trademark of Oracle and/or its affiliates.

Microsoft®, Internet Explorer, Windows®, Windows Server 2007®, Windows XP®, and Windows 7® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Firefox® is a registered trademark of the Mozilla Foundation.

Google Chrome® is a trademark of Google Inc.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

X/Open® is a registered trademark, and the X device is a trademark of X/Open Company Ltd. in the UK and other countries.

Red Hat® is a registered trademark of the Red Hat Company.

Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

Contents

| | |
|---|-----------|
| Preface | 5 |
| Chapter 1..... | 7 |
| The MBBQoE customization | 7 |
| 1.1 Introduction..... | 7 |
| Chapter 2..... | 9 |
| Datasource configuration | 9 |
| Chapter 3..... | 11 |
| Users and roles | 11 |
| 3.1 MBBQoE user and role..... | 11 |
| 3.2 Using roles to control the view | 11 |
| 3.3 Using roles to compute KPIs | 11 |
| Chapter 4..... | 13 |
| File structure..... | 13 |
| 4.1 custom.properties..... | 13 |
| 4.2 configGUIView.xml | 13 |
| 4.2.1 Topbar | 13 |
| 4.2.2 Customer Summary Panel..... | 13 |
| 4.2.3 MBBQoE panel | 14 |
| 4.2.4 Throughput metrics..... | 14 |
| 4.2.5 Customer Quality of Experience..... | 15 |
| 4.2.6 Customer multilevel KPIs | 16 |
| 4.2.7 Using icons in the view | 16 |
| 4.3 Extension.groovy | 17 |
| 4.3.1 Closure definitions..... | 17 |
| 4.3.2 Formatting functions..... | 17 |
| 4.3.3 Computing the experience degradation..... | 18 |
| 4.4 Config.groovy | 18 |
| 4.4.1 Database connection..... | 18 |
| 4.4.2 Admin role | 18 |
| 4.4.3 Msisdn to IMSI conversion | 18 |
| 4.4.4 Customer sessions and throughputs | 19 |
| 4.4.5 Most used cell..... | 20 |
| 4.4.6 Device used..... | 21 |
| 4.4.7 Sub script execution | 22 |
| 4.4.8 Overall statistics..... | 23 |
| 4.4.9 Charts and gauges | 24 |
| 4.4.10 Local functions | 24 |
| 4.5 Config_streaming.groovy | 25 |
| 4.5.1 Zstore tables | 25 |
| 4.5.2 Customer query fields | 25 |

| | | |
|--------|----------------------------------|----|
| 4.5.3 | Customer query | 26 |
| 4.5.4 | Customer KPI computing..... | 27 |
| 4.5.5 | Location query fields..... | 27 |
| 4.5.6 | Location query | 28 |
| 4.5.7 | Location KPI computing..... | 28 |
| 4.5.8 | Device query fields | 28 |
| 4.5.9 | Device query | 30 |
| 4.5.10 | Device KPI computing..... | 30 |
| 4.5.11 | KPI computing | 30 |
| 4.6 | Config_filessharing.groovy | 32 |
| 4.7 | Config_web_browsing.groovy | 32 |
| 4.8 | Config_email.groovy..... | 32 |
| 4.9 | Config_network.groovy | 33 |
| 4.10 | Icons | 35 |
| 4.11 | Messages..... | 35 |

Preface

This guide describes how to configure and customize the Mobile BroadBand Quality of Experience (MBBQoE) customization of the HP Customer Care Dashboard product.

Product Name: Customer Care Dashboard MBBQoE customization

Product Version: V1.0

Kit Version: V1.0-01E

Intended Audience

- This Configuration guide is for anyone who is responsible for customizing the MBBQoE custom.

Software Versions

The term UNIX is used as a generic reference to the operating system, unless otherwise specified.

The software versions referred to in this document are as follows:

| Software | Version | OS | Databases |
|----------|---------|-----------------------------------|--------------------------------------|
| HP CCD | 1.0 | Suse SLES 11 SP1 for x86 machines | SQLite 3.7.2 for management of users |
| HP CEA | 4.5 | Suse SLES 11 SP1 for x86 machines | HP CEA Zstore |

Table 1 - Software versions

Typographical Conventions

Courier Font:

- Source code and examples of file contents.
- Commands that you enter on the screen.
- Pathnames
- Keyboard key names

Italic Text:

- Filenames, programs and parameters.
- The names of other documents referenced in this manual.

Bold Text:

- To introduce new terms and to emphasize important words.

Associated Documents

The following documents contain useful reference information:

- *HP CCD V1.0 - MBBQoE Customization - User Guide*
- *HP CCD V1.0 - Installation Configuration and Administration Guide*
- *HP CCD V1.0 – Release Notes*
- *HP CEA 4.5 Release Notes*
- *HP CEA V4.5 – MBBQoE User Guide*

Support

Please visit our HP Software Support Online Web site at www.hp.com/go/hpsoftwaresupport for contact information, and details about HP Software products, services, and support.

The Software support area of the Software Web site includes the following:

- Downloadable documentation.
- Troubleshooting information.
- Patches and updates.
- Problem reporting.
- Training information.
- Support program information.

The MBBQoE customization

1.1 Introduction

An HP CCD customization is composed, at a minimum, of the following three files:

- configGUIView.xml, which describes the dashboard view presented to the user
- Extension.groovy, which can define global functions used inside the custom. This file may be empty, but needs to exist.
- Config.groovy, which performs all the computations to provide the KPIs that will be displayed in the view

In addition to these three files, the MBBQoE custom contains several other files that will be described in this guide.

Important:

It is important to note that everything in the customization can be modified, even at run time.

Both the XML view and the groovy script are text files, which can be modified using any text editor (including vi).

It is possible to modify the view to change an icon, or to change the threshold values to determine when an indicator displays a red cross or a green check.

It is possible to change the labels by modifying the MessagesBundle.properties files.

It is possible to change the groovy scripts to change the way QoE scores are computed (for instance use a weighted average between all services instead of considering the overall score as the worst of all scores). It is also possible to change the Zsote queries and the KPIs used in the computations.

Whenever any file from the customization is updated, it is immediately taken into account at the next request. There is no need to reinstall a new kit, or restart the server.

All modifications are immediately active.

Warning: always keep a backup copy of all the files before making any modification.

If you introduce errors in the XML, or in the groovy scripts, the application will no longer run, or will show an error.

Groovy scripts are pieces of code, compiled and evaluated at run time. If you

introduce syntax errors in the code, it will no longer compile. If you introduce bugs, it will no longer run.

Datasource configuration

The MBBQoE customization uses one datasource to connect to the HP CEA Zstore database. This datasource is accessed by the following line in Config.groovy:

```
connect "db", "jdbc/Zstore"
```

The datasource must have been created in Tomcat by customizing the `/var/opt/CCD/conf/web.xml` and `/var/opt/CCD/conf/ccd.xml` files on the server as explained in the “HP CCD V1.0 - Installation Configuration and Administration Guide”

In `/var/opt/CCD/conf/web.xml`, add the following lines:

```
<!-- Zstore Datasource -->
<resource-ref>
<description>Zstore Database</description>
<res-ref-name>jdbc/Zstore</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

In Add in `/var/opt/CCD/conf/ccd.xml`, add the following lines:

```
<!-- Zstore Datasource used by the Customer Care
Dashboard for MbbQoe customization -->
<Resource name=\"jdbc/Zstore\" auth=\"Container\"
type=\"javax.sql.DataSource\"
driverClassName=\"com.zhilabs.zstore.jdbc.Driver\"
maxActive=\"100\" maxIdle=\"30\" maxWait=\"10000\"

url=\"jdbc:zstore://YOUR_ZSTORE_IP_ADDRESS:1974//m0/zen5/
zstore-frontend\" username=\"YOUR_ZSTORE_USER\"
password=\"YOUR_ZSTORE_USER_PASSWORD\"/>
```

It is important to use exactly the same string name in `web.xml`, `ccd.xml` and `Config.groovy`.

Alternatively, you can bypass the Tomcat datasource mechanism, by instantiating the connection to the Zstore directly in the `Config.groovy` file, by changing the “connect” line above with the following line:

```
connect "db", "jdbc:zstore://
YOUR_ZSTORE_IP_ADDRESS:1974//m0/zen5/zstore-frontend", "
YOUR_ZSTORE_USER ", " YOUR_ZSTORE_USER_PASSWORD "
```

In both cases, this line establishes a connection to a specific Zstore instance, and associates it to a variable called “db”.

This “db” will be used in the Config to execute queries:

```
read db, cmd
```

If you chose to declare the connection inside Config.groovy, CCD won't be able to use the connection pooling mechanism of Tomcat. This means that for every request, a connection to the Zstore needs to be established, and then released at the end of every request.

In case the KPIs are spread across multiple Zstores, the customization must be updated to point each query to the correct database.

In this case, you must declare one “connect” instruction per Zstore, then modify the queries accordingly:

```
connect "db1" "jdbc/Zstore_Kpis_1"
connect "db2" "jdbc/Zstore_Kpis_2"
...
read db1, cmd
...
read db2, cmd
```

Users and roles

3.1 MBBQoE user and role

Access to CCD customs can be restricted or customized based on user roles.

By default, CCD creates a role named “*admin*”, and a user called “*admin*”, which has the role “*admin*”. Any user having the “*admin*” role can access any custom.

When the MBBQoE customization is installed, it creates a role “*role_mbbqoe*” which can run the *mbbqoe* custom, and a user called “*user_mbbqoe*”, which has the “*role_mbbqoe*”.

When using the *ccd_admin* tool to register new users, ensure that they have the “*role_mbbqoe*” role.

3.2 Using roles to control the view

The file *configGUIView.xml* controls the view that will be presented to the user. It is composed of XML tags like `<column>` or `<widget>` which control how the view is built.

It is possible to include an attribute “`role=<some role>`” to restrict an element to the users who have the specified role.

In the MBBQoE custom, this feature is used to restrict the display of the detailed KPI view only to users with the *admin* role.

This is done through instructions like this one:

```
<widget role="admin" indicator="customer-filesharing-  
service-nonaccessibility"  
title="FileSharing_Service_NonAccessibility"  
width="400px" height="45px" type="Info"/>
```

User_mbbqoe which does not have the *admin* role will not see this widget, or any child widget that might be declared within this element.

It is possible to restrict an element to multiple roles, by using a comma separated list (e.g. `<widget role="role1,role2" ...>`)

3.3 Using roles to compute KPIs

Roles can also be used to control the KPIs computed, or the format of the indicators returned from within *Config.groovy*, by using the keyword “*hasRole*”:

```
if (hasRole("admin")) {  
... Do something  
}
```

```
else
{
... Do something else
}
```

Note that each use of the “hasRole” keyword will trigger a call to the authorization service, so it is recommended to use a global variable if you need to adjust your customization based on a specific role:

```
isAdmin = (hasRole("admin"));
```

In the MBBQoE custom, we use this feature to only display the QoE scores for users who have the admin role:

```
if (isAdmin) {
    kpi "customer-data-overall", customer_dataOverall,
        printPercent(customer_dataOverall)
}
else {
    kpi "customer-data-overall", customer_dataOverall
}
```

File structure

The MBBQoE customization kit is composed of several files, organized in three subfolders, under resources:

- *config*: contains the XML view file and the groovy files.
- *properties*: contains the properties files used to localize the dashboard view
- *icons*: images, icons used in the dashboard view

4.1 custom.properties

This small file contains the name and version of the customization.

```
name=mbbqoe  
version=V1.0
```

The name matches the url parameter used to access the customization (custom=mbbqoe)

4.2 configGUIView.xml

```
<gui xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance"  
xsi:noNamespaceSchemaLocation="configGUIView.xsd"  
class="first-level-customer-care" title="First Level  
Customer Care">
```

This is the top tag of the page

4.2.1 Topbar

```
<topbar></topbar>
```

Placeholder for customization launch links. <launch> tags can be added here

4.2.2 Customer Summary Panel

```
<row class="panel vert-align large">  
  <row class="left" style="width:680px;">  
    <widget width="220px" height="45px"  
type="Info" icon="resources/HP_exp-live/img/icons/HP  
Blue/performance.png" title="Customer"  
indicator="customer"/>  
    <widget width="220px" height="45px"  
type="Info" title = "Location" indicator="location"/>  
    <widget width="220px" height="45px"  
type="Info" title= "Device" indicator="device"/>  
  </row>  
  <row class="right">  
    <widget indicator="apn" title="APN"  
width="280px" height="45px" type="InfoList"
```

```

    icon="resources/HP_exp-live/img/icons/HP
    Blue/graph_72.png"/>
  </row>
</row>

```

This is the Customer Summary Panel, composed of 4 widgets. The two <row> subtags are used to align the widgets either on the left, or right side of the screen.

Reminder: there are different kinds of widgets. The most important attribute is "indicator" which is used to link the widget on the screen, with the KPI computed in Config.groovy. The type attribute can take different values: Info (displays a value), InfoList (displays multiple values in a drop down box), ThresholdIndicator (displays a value with a colored icon indicating OK, KO or Warning), BarChart or Gauge (for graphs)

4.2.3 MBBQoE panel

```

<row>
  <column class="panel" width="42" height="42">
    <!-- =====
    ===== Customer indicators =====
    ===== -->
    ...
  </column>
  <column class="panel middle" >
    <!-- =====
    ===== Location indicators =====
    ===== -->
    ...
  </column>
  <column class="panel right">
    <!-- =====
    ===== Device indicators =====
    ===== -->
    ...
  </column>
</row>

```

The next row is used to organize the next panel in three columns, the customer's KPIs on the left, the Location's KPIs in the middle, and the Device's KPIs on the right.

See below for more details on how each of these columns is organized.

4.2.4 Throughput metrics

```

<row class="panel large vert-align">
  <widget width="110px" height="105px" type="Gauge"
  title="SubscriberSummaryAvgThroughputDown" min="0" max="43008"
  unit="kbps" indicator="throughput_avg_down"/>
  <widget width="110px" height="105px" type="Gauge"
  title="SubscriberSummaryAvgThroughputUp" min="0" max="11264"
  unit="kbps" indicator="throughput_avg_up"/>
  <widget width="110px" height="105px" type="Gauge"
  title="SubscriberSummaryPeakThroughputDown" min="0"
  max="43008" unit="kbps" indicator="throughput_max_down"/>
  <widget width="110px" height="105px" type="Gauge"
  title="SubscriberSummaryPeakThroughputUp" min="0" max="11264"
  unit="kbps" indicator="throughput_max_up"/>
  <column class="right">
    <widget width="200px" height="45px"
    type="Info" icon="resources/HP_exp-live/img/icons/arrow
    up.png" title="SubscriberTotalVolumeUp"
    indicator="total_volume_up"/>
  </column>
</row>

```

```

        <widget width="200px" height="45px"
type="Info" icon="resources/HP_exp-live/img/icons/arrow
down.png" title="SubscriberTotalVolumeDown"
indicator="total_volume_down"/>
    </column>
</row>

```

This panel shows the customer's upload and download throughput, in the form of 4 gauge charts. Each gauge has a max value configured to the theoretical limit of 3G HSDPA (42 Mb for download, 11 Mb for upload).

It is possible to customize the max attribute of the widget, to put values closest to the real experience that we can expect for the users.

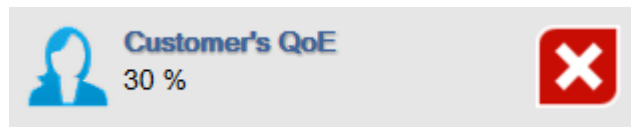
The panel also contains two information widgets showing the total volume of upload and download for the user.

4.2.5 Customer Quality of Experience

```

<widget width="300px" height="45px"
type="ThresholdIndicator" icon="resources/HP_exp-
live/img/icons/HP Blue/user_48.png" title="Customer_QoE"
indicator="customer-overall-status" threshold-inf="50.0"
threshold-sup="70.0"/>

```



This first widget definition corresponds to the global Customer Quality of experience. It has several attributes that determine how it is rendered:

- width, height: determines the size of the panel in the screen
- icon: the image displayed on the left side of the panel (see below for more details on the icons path).
- title: the string is used as a key in the properties file MessagesBundle_en.properties. If the key is found, then the widget's title will be replaced by the value in the properties file. If the key does not exist, then the title of the widget will be the value of the title attribute.

If the user has specified a locale other than English, and there is a file MessagesBundle_<locale>.properties, then the key will be looked-up into the localized file.

In MessagesBundle_en.properties:

```
Customer_QoE = Customer's QoE
```

In MessagesBundle_fr.properties:

```
Customer_QoE = QdE Utilisateur
```

- indicator: the name of KPI indicator to display. This indicator will be computed during execution of the Config.groovy file.
- type: ThresholdIndicator means that the indicator will be displayed with an icon either green (status OK), yellow (warning) or red (status is bad). The icon displayed depends on the indicator's score, and the two following thresholds.
- threshold-inf, threshold-sup: Two values that determine which icon will be used to show the quality score. If the indicator value is lower than threshold-inf (here 50%), then the "Red cross" icon will be used. If the indicator value is greater than threshold-sup (70%), then the "green

check” icon will be used. If the value is between these two thresholds, then the “yellow warning” icon will be used.

4.2.6 Customer multilevel KPIs

```
<widget indicator="location-streaming"
title="Streaming_QoE_Score" ...>
  <widget indicator="location-streaming-acc"
title="Streaming_Accessibility" ...>
    <widget role="admin" indicator="location-
streaming-repro-start-failure-ratio" .../>
    <widget role="admin" indicator="location-
streaming-service-nonaccessibility" ... />
  </widget>
  <widget indicator="location-streaming-retain"
title="Streaming_Retainability" ... >
    <widget role="admin" indicator="location-
streaming-rebuff-failure-ratio"... />
    <widget role="admin" indicator="location-
streaming-repro-cutoff-ratio" ... />
  </widget>
  <widget indicator="location-streaming-qual" ...>
    <widget role="admin" indicator="location-
streaming-rebuff-time-pct" ... />
    <widget role="admin" indicator="location-
streaming-time-to-stream-start"... />
  </widget>
</widget>
```

This (simplified) code snippet show the three-level widget panel used to display the scores for each service, then per category.

The first <widget> on top corresponds to the Streaming service. Within the <widget> element, we can find three other <widget> elements, corresponding to the categories “Accessibility”, “Retainability” and “Quality”.

Because the <widget> elements are declared inside the parent’s <widget> element (between the <widget> and </widget> tags), and not alongside the parent, they will be automatically displayed in a sub-panel, when the user clicks on the parent. An arrow is displayed to let the user know that more detail is available.

Inside the “Accessibility” widget, there are two more <widget> defined, corresponding to the detailed KPIs used to compute the quality score.

These widgets have the attribute role=”admin”, which means that only a user with the admin role would see these widgets. For any other user, it is as if these elements did not exist.

4.2.7 Using icons in the view

Each <widget> can have an “icon” attribute , containing the relative path to an image file.

HP CCD provides a number of predefined icons from the HP Experience Live library.

This icon can be referenced via the path “resources/HP_exp-live”:

```
icon="resources/HP_exp-live/img/icons/arrow up.png"
```

You can also reference icons included inside the MBBQoE customization, in the folder resources/icons.

To reference these icons, the path to use is “resources/mbbqoe/” (where mbbqoe is the name of the deployed customization):

```
icon="resources/mbbqoe/File_Sharing_sharing_RGB_gray_48.png"
```

4.3 Extension.groovy

This file is used to extend the DSL language, by creating new keywords, or it can also be used to define global variable or global functions used in the execution of Config.groovy.

It is possible to create functions in Config.groovy, however, these functions can only be used inside the file where they are created. For global functions, Extension.groovy must be used.

4.3.1 Closure definitions

```
degradation = {Double value, Double threshold ->
computeDegradation(value, threshold, 0,
DEGRADATION_DEFAULT)}
printPercent = {Double value -> printPercent(value)}
printKbps = {Double value -> printKbps(value)}
printKbytes = {Double bytes -> printKbytes(bytes)}
printMbytes = {Double bytes -> printMbytes(bytes)}
printSeconds = {Double value -> printSeconds(value)}
nonnull = {Double value -> nonnull(value)}
displayKpiDegradationPct = {Double value, Double
threshold -> displayKpiDegradationPct(value, threshold)}
displayKpiDegradationSeconds = {Double value, Double
threshold -> displayKpiDegradationSeconds(value,
threshold)}
```

In order to access a function created in Extension.groovy from the Config.groovy file, it must be declared as a closure, in the form

```
<keyword> = { <input parameters> -> <function name>(parameters) }
```

4.3.2 Formatting functions

The following functions are provided to provide a consistent way to display QoE scores:

- **printPercent**: displays the value rounded to the nearest integer, with a % sign
- **printKbps**: Displays a value (containing bps) as kbps
- **printKbytes**: Displays a value (containing bytes) as kb
- **printMbytes**: Displays a value (containing bytes) as Mb
- **printSeconds**: Displays a value (containing ms) as seconds
- **nonnull**: converts null values to 0
- **displayKpiDegradationPct**: displays a string showing the KPI score, the threshold (in percentage), and the computed degradation based on this threshold (e.g.: 1 % (threshold:5 %) Service Degradation:1 % QoE:99 %)
- **displayKpiDegradationSeconds**: displays a string showing the KPI score, the threshold (in seconds), and the computed degradation based on this

```
threshold (e.g:0.0 s (threshold:0.5 s) Service
Degradation:0 % QoE:100 %)
```

4.3.3 Computing the experience degradation

The `computeDegradation` function determines the performance degradation of a service, by comparing the KPI value to a predefined threshold.

- At 50% of the value of the threshold, there is a degradation factor already of 25% (QoE Score)
- At 100% of threshold, the degradation factor is of 50%
- Between 100% of the threshold and 200% of the threshold, the degradation evolves from 50% to 75%
- Between 200% of the threshold and 400% of the threshold, the degradation varies from 75% to 100%
- When the value goes beyond 400% of the threshold, the degradation is 100%
- For values between 0% and 50% of the threshold, the degradation factor will vary from 0% to 25%. However, this degradation is not linear. Instead, we use a formula based on Arctangent which will give a lower degradation factor for values closest to 0%, and higher degradation as values reach closer to 50% (lower and higher compared to what a linear formula would return).

4.4 Config.groovy

4.4.1 Database connection

```
connect "db", "jdbc/Zstore"
```

This line assigns to variable “db” the datasource connection named “jdbc/Zstore” (which must be configured in Tomcat).

“db” will be used in all subsequent queries, to read data from the Zstore

4.4.2 Admin role

```
isAdmin = (hasRole("admin"));
```

Calls the “hasRole” service, to determine if the user has the “admin” role. The Boolean result is assigned to variable “isAdmin”.

It is important to note here that “isAdmin” is not defined through a “def isAdmin” command.

When declaring a variable with “def”, you create a local variable, usable only in the context of the script (here Config.groovy). However, when the variable is created without the “def” keyword, it becomes a global variable, usable during the whole execution, both of the current script, and all sub-scripts.

In this case, we want the “isAdmin” Boolean to be usable in all the groovy subscripts called by Config.groovy, which is the reason why “def” is not used.

4.4.3 Msisdn to IMSI conversion

1. `read db, concat("SELECT FIELD:BEARER.IMSI FROM index.msisdn-imsi-index WHERE BEARER.MSISDN = '", escapeString(customerId), "'")`
2. `def imsi = getVariable("field:bearer.imsi")`

```

3. if (imsi != null && imsi.size() > 0 && imsi[0] !=
    null)
4. {
        log.debug(concat("MSISDN [", customerId, "] -
        > IMSI [", imsi[0], "]"));
        customerId = imsi[0]
5. }

6. info "customer", customerId

```

Line [1] executes a direct query to the Zstore. The user has entered a value in the customerId field, and we need to determine if this value is an MSISDN, or an IMSI,

The query searches for all records in table “index.msisdn-imsi-index” if there is an MSISDN equal to the provided id.

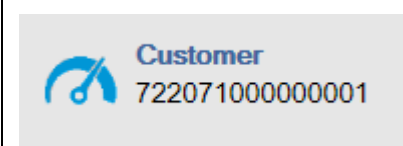
Line [2]: when executing a query to the Zstore for field X, the resulting recordset will be returned in a variable that can be accessed via the instruction “getVariable(‘x’)”. It is important to note that even if the Select statement can use any casing for the field name (here FIELD:BEARER.IMSI in all uppercase), the variable MUST use the same name in all lowercase to access the variable (here field:bearer.imsi).

Line [3-7]: All queries return a list of records (for 0 to n), so all field variables are returned as arrays of size 0 to n.

In this query, we expect the MSISDN to IMSI query to return either 0 or 1 record. So if we have more than 0 record, then we know that the provided id is a MSISDN, so we assign to the global customerId variable the corresponding IMSI (line [6]).

If the query did not return any record, then we keep the customerId as it was provided by the user.

Line [8]: *info* is a keyword used assign a text string to a widget in the view. Here, we assign the customerId value (the IMSI) to the widget called “customer” in configGUIView.xml. The result will be:

| | |
|---|--|
|  | <pre> <widget width="220px" height="45px" type="Info" icon="resources/HP_exp- live/img/icons/HP Blue/performance.png" title="Customer" indicator="customer"/> </pre> |
|---|--|

4.4.4 Customer sessions and throughputs

```

1. read (db, "index.correlator-grid", sql_olap_dimension,
    "FIELD:BEARER.APN", "FIELD:BEARER.CELL",
    "field:location.mcc", "field:location.mnc",
    "field:location.lac", "field:location.sac",
    "field:bearer.imeisv", "field:bearer.summary-average-
throughput-down", "field:bearer.summary-average-throughput-
up", "field:bearer.summary-throughput-down",
    "field:bearer.summary-throughput-up", "field:bearer.total-
volume-up", "field:bearer.total-volume-down")

2. def apn = simplify(getVariable("field:bearer.apn"))
3. info "apn", listToInfoListDisplay(apn)

4. // Unit is bps for throughput indicators, bytes for volume
5. def subscriberSummaryAvgThroughputDown =
    nonnull (avg (getVariable ("field:bearer.summary-average-
throughput-down")))

```

6. `def subscriberSummaryAvgThroughputUp =
nonnull (avg (getVariable ("field:bearer.summary-average-
throughput-up")))`
7. `def subscriberSummaryPeakThroughputDown =
nonnull (max (getVariable ("field:bearer.summary-throughput-
down")))`
8. `def subscriberSummaryPeakThroughputUp =
nonnull (max (getVariable ("field:bearer.summary-throughput-
up")))`
9. `def subscriberTotalVolumeDown =
nonnull (sum (getVariable ("field:bearer.total-volume-down")))`
10. `def subscriberTotalVolumeUp =
nonnull (sum (getVariable ("field:bearer.total-volume-up")))`

[1]. In the previous paragraph, we saw that the “read” command could have the syntax “read db, cmd”, to execute any command to the database. The other syntax of the “read” command is used here:

```
read db, table, field1, ..., fieldn
```

When using this syntax, a Zstore query is built to return fields 1-n from the specified table, for all records in the selected time period, and for the specified customerId. In Zstore syntax, there is a “RANGE KEYJOIN” criteria added to the query. This is hidden from view to make this script easier to read.

So here, this query will return a number of fields for all records from the *index.correlator-grid* table for the range period considered.

[2-3]. As seen previously, each field returned by the query can be obtained with the `getVariable` function, as an array of values. The *simplify* keyword will remove from the array all duplicate values, and we use the *info* keyword to assign the list of APN values to the display widget.

The APN widget is an `InfoList`, which means that it can display multiple values in a drop down (`listToInfoListDisplay` is a function that converts an array of values to the correct format to be displayed in an `InfoList`)

| | |
|---|---|
|  | <pre><widget indicator="apn" title="APN" width="280px" height="45px" type="InfoList" icon="resources/HP_exp- live/img/icons/HP Blue/graph_72.png"/></pre> |
|---|---|

[5-10]: The query may have returned multiple records, one per every session for the customer during the time period. For each of these sessions, we have retrieved information corresponding to the average and max throughput *during the session*, as well as the total volume up/downloaded *during the session*.

In order to display these indicators for the whole period, the script uses the keywords “sum”, “avg” or “max” to compute the total, max or average throughputs for the user during the whole period.

4.4.5 Most used cell

1. `def cell = most (getVariable ("field:bearer.cell"))`
2. `def locations = buildLocationString()`

```

3. def location = most(locations);

4. cellId = null; // cellID need to be defined as a global
   variable

5. def locationDisplay = "";
6. if (location != null)
7. {
8.   def cmd = concat("SELECT INDEX:LOCATION.COORDS FROM
   index.mcc-mnc-lac-sac-cid-location-index ", " WHERE
   LOCATION.MCC-MNC-LAC-SAC-CID = '", escapeString(location),
   "'")
9.   read db, cmd
10.  def coords = getVariable("index:location.coords");
11.  log.debug(cmd + " --> " + coords);
12.  def coord = most(coords);
13.  cellId = getCellIdFromCoords(coord);
14.  locationDisplay = getLocationDisplayString(coord,
   cellId);
15. }
16. info "location", locationDisplay

```

[1-3] we have a number of records corresponding to all the sessions of the user during the time period. Each of these sessions is linked to one cell. To get the id of the most used cell, we need to build the unique key corresponding to a location. This key is a composite string built from multiple fields, mcc, mnc, lac and sac. The function “buildLocationString” generates these composite keys.

The “most” keyword will return the value that appears most in the list of values. In this case, this is the key of the cell that was used during the most number of sessions.

[8-12] Execute a query on table index.mcc-mnc-lac-sac-cid-location-index to get the location coordinates for the most used cell (based on the mcc-mnc-lac-sac key).

If the query returns a value, it will be in the form “<town>|||<cellid>;<coordinates>” (e.g. Chaco|||VHE002;-27.425862;-59.031322;5)

[13] The cellId is the key that must be used as a RangeKeyJoin is all the Zstore “cell” tables. It is the element of the coordinates string between ||| and ;

This id is stored in variable “cellId”. Note again that this variable is not declared with “def”, as it needs to be global to be used in the sub-scripts.

[14, 16] Builds the location display string that will be assigned to the view widget. The function takes into account the user’s role and only shows the town for admin users

4.4.6 Device used

```

1. def imeisv = most(getVariable("field:bearer.imeisv"))
2. def tac = (imeisv == null ? null : imeisv.substring(0, 8))

3. def deviceModel = null;
4. def deviceBrand = null;
5. def deviceOS = null;
6. def deviceOSVersion = null;
7. deviceBMO = null; // deviceBMO need to be defined as a global
   variable

8. if (tac != null)
9. {

```

```

10.     def cmd = concat("SELECT FIELD:DEVICE.MODEL,
FIELD:DEVICE.BRAND, FIELD:DEVICE.OS, FIELD:DEVICE.OS-VERSION",
" FROM index.imei-index WHERE imei.tac = '", escapeString(tac)
+ "'")
11.     read db, cmd
12.     deviceModel =
noquotes(most(getVariable("field:device.model")))
13.     deviceBrand =
noquotes(most(getVariable("field:device.brand")))
14.     deviceOS = noquotes(most(getVariable("field:device.os")))
15.     deviceOSVersion =
noquotes(most(getVariable("field:device.os-version")))
16.     deviceBMO = concat(deviceBrand, "|||", deviceModel,
"|||", deviceOS)
17.     }
18.     def deviceDisplay = getDeviceDisplayString(deviceModel,
deviceBMO, deviceBrand, deviceOS, deviceOSVersion)
19.     info "device", deviceDisplay

```

[1-2] From the list of all user sessions, we obtain the list of all “imeisv” used by the customer, we select the “most used”, and from this imeisv, we extract the tac number, which is the first 8 characters of the imeisv.

[10-16] This “tac” is used as a key to query table `index.imei-index`, and get back the device brand, model and OS. These three fields are concatenated in a single string called DeviceBMO, which will be used as the key to get all KPIs in the “device” tables of the Zstore.

Again, deviceBMO is not declared with “def” to make it global.

4.4.7 Sub script execution

```

1. //----- Web Browsing Kpis -----
2. customer_web_qoe = null
3. location_web_qoe = null;
4. device_web_qoe = null;
5. evaluate "Config_web_browsing.groovy"
6. //----- end of Web Browsing Kpis -----

7. //----- Streaming Kpis -----
8. customer_streaming_qoe = null
9. location_streaming_qoe = null;
10. device_streaming_qoe = null;
11. evaluate "Config_streaming.groovy"
12. //----- end of Streaming Kpis -----

13. //----- FileSharing Kpis -----
14. customer_filesharing_qoe = null
15. location_filesharing_qoe = null;
16. device_filesharing_qoe = null;
17. evaluate "Config_filesharing.groovy"
18. //----- end of FileSharing Kpis -----

19. //----- Email Kpis -----
20. customer_email_qoe = null
21. location_email_qoe = null;
22. device_email_qoe = null;
23. evaluate "Config_email.groovy"
24. //----- end of Email Kpis -----

25. //----- Network Kpis -----

```

```

26. log.debug("%%%%%%%%%%%% Evaluate Config_network ")
27. customer_network_qoe = null;
28. location_network_qoe = null;
29. device_network_qoe = null;
30. evaluate "Config_network.groovy"
31. //----- end of Network Kpis -----

```

For readability and maintainability, the main KPI computations have been split into multiple sub-scripts.

[1-6] The “evaluate” keyword triggers the execution of a Groovy script, here `Config_web_browsing.groovy`. This subscript will be executed in its own context, without having access to the variables defined in `Config.groovy`.

The subscript will only be able to access global functions (declared in `Extension.groovy`), and global variables (declared in `Config.groovy` without the `def` keyword).

The subscript will compute many KPIs and update the associated widgets. However, there are 3 qoe scores that will be used in the `Config.groovy` script to compute the overall Qoe score. This is the reason why `customer_web_qoe`, `location_web_qoe` and `device_web_qoe` are initialized (to null) in this script. To make these variables global, so they can be updated in a subscript, while being also available in the main parent script.

[7-12]. same principle. Calls a subscript to compute all the streaming KPIs

[13-18]. same principle. Calls a subscript to compute all the file-sharing KPIs

[19-24]. same principle. Calls a subscript to compute all the email KPIs

[25-31. same principle. Calls a subscript to compute all the network KPIs

4.4.8 Overall statistics

```

1. def customer_dataOverall = min(customer_streaming_qoe,
   customer_filesharing_qoe, customer_web_qoe, customer_email_qoe)
2. def location_dataOverall = min(location_streaming_qoe,
   location_filesharing_qoe, location_web_qoe, location_email_qoe)
3. def device_dataOverall = min(device_streaming_qoe,
   device_filesharing_qoe, device_web_qoe, device_email_qoe)

4. if (isAdmin) {
   kpi "customer-data-overall", customer_dataOverall,
   printPercent(customer_dataOverall)
   kpi "location-data-overall", location_dataOverall,
   printPercent(location_dataOverall)
   kpi "device-data-overall", device_dataOverall,
   printPercent(device_dataOverall)
5. }
6. else {
   kpi "customer-data-overall", customer_dataOverall
   kpi "location-data-overall", location_dataOverall
   kpi "device-data-overall", device_dataOverall
7. }

8. /**
9. * Total QoE Score %      Min (Data Services QoE Score, Network
   QoE Score)
10. */
11. def customer_overall = overallMin ([customer_dataOverall,
   customer_network_qoe])
12. def location_overall = overallMin ([location_dataOverall,
   location_network_qoe])

```

```

13.     def device_overall = overallMin ([device_dataOverall,
device_network_qoe])
14.     if (isAdmin) {
kpi "customer-overall-status", customer_overall,
printPercent(customer_overall)
kpi "location-overall-status", location_overall,
printPercent(location_overall)
kpi "device-overall-status", device_overall,
printPercent(device_overall)
15.     }
16.     else
17.     {
kpi "customer-overall-status", customer_overall
kpi "location-overall-status", location_overall
kpi "device-overall-status", device_overall
18.     }

```

[1] computes the customer's Data Services Overall Qoe, which is the minimum between customer's streaming, web browsing, file sharing and email QoEs.

[2-3] same as above, for the location's and device's Data Services Overall QoE.

[4-7] the QoE scores are assigned to the KPIs, so that the widget displays the right icon, based on the score. If the user has the admin role (line [4]), then in addition to setting the KPI value, we add a label to the KPI showing the score as a percentage. If the user does not have the admin role (line [6]), the percentage won't be displayed.

[10-18] The Overall QoE scores are computed for customer, location and device. The overall score is the min between the Data Service Overall QoE, and the network QoE.

4.4.9 Charts and gauges

```

1. // convert throughput from bps to kbps and volume from bytes to Mb
2. kpi "throughput_avg_down", subscriberSummaryAvgThroughputDown /
1024
3. kpi "throughput_avg_up", subscriberSummaryAvgThroughputUp / 1024
4. kpi "throughput_max_up", subscriberSummaryPeakThroughputUp / 1024
5. kpi "throughput_max_down", subscriberSummaryPeakThroughputDown /
1024
6. info "total_volume_down", printMbytes(subscriberTotalVolumeDown)
7. info "total_volume_up", printMbytes(subscriberTotalVolumeUp)

```

This code assigns the upload/download throughput values to their respective KPIs, so they can be rendered.

The difference between keywords kpi and info, is that kpi assigns a value that the widget will have to render (in this case, the values are rendered in gauge charts).

On the other hand, info sets a text string, that will typically be rendered in an Info widget.

4.4.10 Local functions

Config.groovy contains a number of functions that are used in the processing. These functions are defined in this file, and not in Extension.groovy because they are exclusively used by the Config.groovy script, so they don't need to be global.

- escapeString: escapes quotes from any parameter used in queries to prevent SQL injection
- buildLocationString: builds location strings based on mcc-mnc-sac-lac

- `getCellIdFromCoords`: extracts the `cellId` from a coordinates string
- `getTownFromCoords`: extracts the town from a coordinates string
- `getLocationDisplayString`: determines the format of the location to display, based on the user's role
- `getDeviceDisplayString`: determines the format of the device info to display
- `listToInfoListDisplay`: converts an array of values, into a string that can be rendered by an `InfoList` widget
- `overallMin`: computes the minimum of a list of values, without failing if some values are null.

4.5 Config_streaming.groovy

This script contains all the queries and computations used to determine the streaming KPIs, and update the widgets.

The formulas to determine QoE scores per KPI and per service are the same for the customer, the location and the device. The difference lies in the table and field names to query to retrieve customer data, location data, or device data.

This is why the first section of the script builds a query to retrieve data from the customer data, then calls a procedure to compute the customer KPIs.

This same procedure is called again to compute the location KPIs after we have built another query to get location data, and finally a third time with the device data.

4.5.1 Zstore tables

```
def subscriberStreamingTable = "report.gn_pdu_streaming-
subscriber-900-timeline"
def locationStreamingTable = "report.gn_pdu_streaming-cell-
900-timeline"
def deviceStreamingTable = "report.gn_pdu_streaming-
brand_model_os-900-timeline"
```

These are the tables containing the KPI data for customer, location and device.

4.5.2 Customer query fields

```
// Customer KPIs
def sql_st_cutoff = "[min(100,
100*safediv((1*#{gn_pdu_streaming-subscriber-900-timeline-
common.count-streaming-failure-count-3:gn_pdu_streaming-
subscriber-900-timeline-common.count-streaming-failure-count-
3}),1*#{gn_pdu_streaming-subscriber-900-timeline-common.count-
streaming-session-count-2:gn_pdu_streaming-subscriber-900-
timeline-common.count-streaming-session-count-2}), 1))]" as
streaming_Repro_Cutoff_Ratio"
def sql_st_failure = "[min(100,
100*safediv((1*#{gn_pdu_streaming-subscriber-900-timeline-
streaming_rebuffering_failure_ratio.count-streaming-failure-
count-1:gn_pdu_streaming-subscriber-900-timeline-
streaming_rebuffering_failure_ratio.count-streaming-failure-
count-1}), 1*#{gn_pdu_streaming-subscriber-900-timeline-
common.count-streaming-session-count-2:gn_pdu_streaming-
subscriber-900-timeline-common.count-streaming-session-count-
2}), 1))]" as streaming_Rebuff_Failure_Ratio"
def sql_st_timepct = "[safediv(100*(#{gn_pdu_streaming-
subscriber-900-timeline-common.average-tcp-rebuffering-time-
```

```

1:gn_pdu_streaming-subscriber-900-timeline-common.average-tcp-
rebuffering-time-1}},{gn_pdu_streaming-subscriber-900-
timeline-streaming_rebuffering_time_percentage.average-stream-
duration-1:gn_pdu_streaming-subscriber-900-timeline-
streaming_rebuffering_time_percentage.average-stream-duration-
1}}] as streaming_Rebuff_Time_Pct"
def sql_st_startfail = "[min(100,
100*safediv((1*#{gn_pdu_streaming-subscriber-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-1:gn_pdu_streaming-subscriber-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-1})+1*#{gn_pdu_streaming-subscriber-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-2:gn_pdu_streaming-subscriber-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-2})+1*#{gn_pdu_streaming-subscriber-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-3:gn_pdu_streaming-subscriber-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-3}),1*#{gn_pdu_streaming-subscriber-900-
timeline-common.count-streaming-session-count-
1:gn_pdu_streaming-subscriber-900-timeline-common.count-
streaming-session-count-1}), 1))]] as
streaming_Repro_Start_Failure"
def sql_st_nonacc = "[min(100,
100*safediv(1*#{gn_pdu_streaming-subscriber-900-timeline-
common.count-streaming-failure-count-3:gn_pdu_streaming-
subscriber-900-timeline-common.count-streaming-failure-count-
3}),1*#{gn_pdu_streaming-subscriber-900-timeline-common.count-
streaming-session-count-1:gn_pdu_streaming-subscriber-900-
timeline-common.count-streaming-session-count-1}), 1))]] as
streaming_Service_Non_Acc"
def sql_st_timetostart = "[#{gn_pdu_streaming-subscriber-900-
timeline-common.average-tcp-service-start-time-
3:gn_pdu_streaming-subscriber-900-timeline-common.average-tcp-
service-start-time-3})+#{gn_pdu_streaming-subscriber-900-
timeline-common.average-http-setup-time-2:gn_pdu_streaming-
subscriber-900-timeline-common.average-http-setup-time-2}}] as
streaming_Time_To_Stream_Start"

```

Each of these commands declares a Zstore formula to return a computation as if it was a Zstore field. It should be noted that these formulas end with “as <something>”, which means that the resulting data will be automatically assigned to the variable of the same name.

So in order to use the data that was returned “as `streaming_Time_To_Stream_Start`”, it is not necessary to use `getVariable(“streaming_time_to_stream_start”)`, we can use `streaming_time_to_stream_start` (name should always be lowercase)

4.5.3 Customer query

```

read (db, subscriberStreamingTable, sql_olap_dimension,
sql_st_cutoff, sql_st_failure, sql_st_timepct,
sql_st_startfail, sql_st_nonacc, sql_st_timetostart);

```

Makes the query to the Zstore to retrieve all the fields (formulas) specified above, from the customer’s table.

Note that due to limitations with the Zstore database, it is recommended when querying multiple fields to always start with field “OLAP.DIMENSION”.

The “read” keyword used in this way will automatically add to the request the RANGE KEYJOIN based on the customer Id and the time period.

4.5.4 Customer KPI computing

```
customer_streaming_qoe = buildStreamingKpis("customer-");
```

Calls the buildStreamingKpis function, with the prefix “customer”. The function will use the data returned by the query, and compute a number of KPIs. By convention, the widgets attached to the customer KPIs all start with ‘customer-’.

The function returns an overall QoE score, which is assigned to customer_streaming_qoe. This is a global variable which was declared in the parent script Config.groovy.

4.5.5 Location query fields

```
// Location KPIs
location_streaming_qoe = null;
if (cellId != null)
{
    sql_st_cutoff = "[min(100,
100*safediv((1*#{gn_pdu_streaming-cell-900-timeline-
common.count-streaming-failure-count-3:gn_pdu_streaming-cell-
900-timeline-common.count-streaming-failure-count-
3}),1*#{gn_pdu_streaming-cell-900-timeline-common.count-
streaming-session-count-2:gn_pdu_streaming-cell-900-timeline-
common.count-streaming-session-count-2}, 1))] as
streaming_Repro_Cutoff_Ratio"
    sql_st_failure = "[min(100,
100*safediv((1*#{gn_pdu_streaming-cell-900-timeline-
streaming_rebuffering_failure_ratio.count-streaming-failure-
count-1:gn_pdu_streaming-cell-900-timeline-
streaming_rebuffering_failure_ratio.count-streaming-failure-
count-1}), 1*#{gn_pdu_streaming-cell-900-timeline-
common.count-streaming-session-count-2:gn_pdu_streaming-cell-
900-timeline-common.count-streaming-session-count-2}, 1))] as
streaming_Rebuff_Failure_Ratio"
    sql_st_timepct = "[safediv(100*(#{gn_pdu_streaming-cell-
900-timeline-common.average-tcp-rebuffering-time-
1:gn_pdu_streaming-cell-900-timeline-common.average-tcp-
rebuffering-time-1}),#{gn_pdu_streaming-cell-900-timeline-
streaming_rebuffering_time_percentage.average-stream-duration-
1:gn_pdu_streaming-cell-900-timeline-
streaming_rebuffering_time_percentage.average-stream-duration-
1})) as streaming_Rebuff_Time_Pct"
    sql_st_startfail = "[min(100,
100*safediv((1*#{gn_pdu_streaming-cell-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-1:gn_pdu_streaming-cell-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-1}+1*#{gn_pdu_streaming-cell-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-2:gn_pdu_streaming-cell-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-2}+1*#{gn_pdu_streaming-cell-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-3:gn_pdu_streaming-cell-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-3}),1*#{gn_pdu_streaming-cell-900-timeline-
```

```

common.count-streaming-session-count-1:gn_pdu_streaming-cell-
900-timeline-common.count-streaming-session-count-1}, 1))]] as
streaming_Repro_Start_Failure"
    sql_st_nonacc = "[min(100,
100*safediv(1*#{gn_pdu_streaming-cell-900-timeline-
common.count-streaming-failure-count-3:gn_pdu_streaming-cell-
900-timeline-common.count-streaming-failure-count-
3},1*#{gn_pdu_streaming-cell-900-timeline-common.count-
streaming-session-count-1:gn_pdu_streaming-cell-900-timeline-
common.count-streaming-session-count-1}, 1))]] as
streaming_Service_Non_Acc"
    sql_st_timetostart = "[#{gn_pdu_streaming-cell-900-
timeline-common.average-tcp-service-start-time-
3:gn_pdu_streaming-cell-900-timeline-common.average-tcp-
service-start-time-3}+#{gn_pdu_streaming-cell-900-timeline-
common.average-http-setup-time-2:gn_pdu_streaming-cell-900-
timeline-common.average-http-setup-time-2}]] as
streaming_Time_To_Stream_Start"

```

Here are the formulas to extract the location KPIs from the Zstore cell table.

It looks very similar to the equivalent formulas used to get back customer KPIs, except for the fact that these fields would contain the string “**_pdu_streaming-cell-900-timeline**” while the customer fields had the string “**_pdu_streaming-subscriber-900-timeline**”.

However, even if the fields used in the formulas are different, the important point is that the “as” part assigns the resulting recordsets to the same variable (**streaming_time_to_stream_start**). This is how the buildStreamingKpis function works, by relying on the “as variables”.

The data is read from different fields and different tables, but it is stored in the same variables, and this is from where the KPIs are computed.

4.5.6 Location query

```

readwhere (db, locationStreamingTable, cellId,
sql_olap_dimension, sql_st_cutoff, sql_st_failure,
sql_st_timepct, sql_st_startfail, sql_st_nonacc,
sql_st_timetostart);

```

The “readwhere” keyword works in a similar way to “read”. The only difference is an extra parameter after the table name. Here, cellId.

The syntax is

```
Readwhere db, table, id, field1... fieldn
```

It will query all the fields from the Zstore table, by adding a RANGE KEYJOIN based on the time period, plus the specified id field (where “read” was automatically using the customerId in the Range KeyJoin)

4.5.7 Location KPI computing

```
location_streaming_qoe = buildStreamingKpis("location-");
```

Calls the buildStreamingKpis function, with the prefix “location-”. The function returns an overall QoE score, which is assigned to location_streaming_qoe. This is a global variable which was declared in the parent script Config.groovy.

4.5.8 Device query fields

```
// Device KPIs
```

```

device_streaming_goe = null;
if (deviceBMO != null)
{
    sql_st_cutoff = "[min(100,
100*safediv((1*#{gn_pdu_streaming-brand_model_os-900-timeline-
common.count-streaming-failure-count-3:gn_pdu_streaming-
brand_model_os-900-timeline-common.count-streaming-failure-
count-3}),1*#{gn_pdu_streaming-brand_model_os-900-timeline-
common.count-streaming-session-count-2:gn_pdu_streaming-
brand_model_os-900-timeline-common.count-streaming-session-
count-2}, 1))] as streaming_Repro_Cutoff_Ratio"
    sql_st_failure = "[min(100,
100*safediv((1*#{gn_pdu_streaming-brand_model_os-900-timeline-
streaming_rebuffering_failure_ratio.count-streaming-failure-
count-1:gn_pdu_streaming-brand_model_os-900-timeline-
streaming_rebuffering_failure_ratio.count-streaming-failure-
count-1}), 1*#{gn_pdu_streaming-brand_model_os-900-timeline-
common.count-streaming-session-count-2:gn_pdu_streaming-
brand_model_os-900-timeline-common.count-streaming-session-
count-2}, 1))] as streaming_Rebuff_Failure_Ratio"
    sql_st_timepct = "[safediv(100*(#{gn_pdu_streaming-
brand_model_os-900-timeline-common.average-tcp-rebuffering-
time-1:gn_pdu_streaming-brand_model_os-900-timeline-
common.average-tcp-rebuffering-time-1}),#{gn_pdu_streaming-
brand_model_os-900-timeline-
streaming_rebuffering_time_percentage.average-stream-duration-
1:gn_pdu_streaming-brand_model_os-900-timeline-
streaming_rebuffering_time_percentage.average-stream-duration-
1}]] as streaming_Rebuff_Time_Pct"
    sql_st_startfail = "[min(100,
100*safediv((1*#{gn_pdu_streaming-brand_model_os-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-1:gn_pdu_streaming-brand_model_os-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-1})+1*#{gn_pdu_streaming-brand_model_os-900-
timeline-streaming_reproduction_start_failure_ratio.count-
streaming-failure-count-2:gn_pdu_streaming-brand_model_os-900-
timeline-streaming_reproduction_start_failure_ratio.count-
streaming-failure-count-2})+1*#{gn_pdu_streaming-
brand_model_os-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-3:gn_pdu_streaming-brand_model_os-900-timeline-
streaming_reproduction_start_failure_ratio.count-streaming-
failure-count-3}),1*#{gn_pdu_streaming-brand_model_os-900-
timeline-common.count-streaming-session-count-
1:gn_pdu_streaming-brand_model_os-900-timeline-common.count-
streaming-session-count-1}, 1))] as
streaming_Repro_Start_Failure"
    sql_st_nonacc = "[min(100,
100*safediv(1*#{gn_pdu_streaming-brand_model_os-900-timeline-
common.count-streaming-failure-count-3:gn_pdu_streaming-
brand_model_os-900-timeline-common.count-streaming-failure-
count-3}),1*#{gn_pdu_streaming-brand_model_os-900-timeline-
common.count-streaming-session-count-1:gn_pdu_streaming-
brand_model_os-900-timeline-common.count-streaming-session-
count-1}, 1))] as streaming_Service_Non_Acc"
    sql_st_timetostart = "[#{gn_pdu_streaming-brand_model_os-
900-timeline-common.average-tcp-service-start-time-
3:gn_pdu_streaming-brand_model_os-900-timeline-common.average-
tcp-service-start-time-3})+#{gn_pdu_streaming-brand_model_os-
900-timeline-common.average-http-setup-time-

```

```
2:gn_pdu_streaming-brand_model_os-900-timeline-common.average-  
http-setup-time-2}} as streaming_Time_To_Stream_Start"
```

4.5.9 Device query

```
readwhere (db, deviceStreamingTable, deviceBMO,  
           sql_olap_dimension, sql_st_cutoff, sql_st_failure,  
           sql_st_timepct, sql_st_startfail, sql_st_nonacc,  
           sql_st_timetostart);
```

This is identical to the location section. The difference is that the table name (and the table fields) are “streaming-brand_model_os” instead of “streaming-cell”.

And the id used for the query Range KeyJoin is the Device BMO string (concatenation of brand, model and os).

4.5.10 Device KPI computing

```
device_streaming_qoe = buildStreamingKpis("device-");
```

Calls the buildStreamingKpis function, with the prefix “device-”. The function returns an overall QoE score, which is assigned to device_streaming_qoe. This is a global variable which was declared in the parent script Config.groovy.

4.5.11 KPI computing

```
if (streaming_repro_start_failure.size() == 0)  
{  
    return null;  
}
```

Tests the number of records that were returned by the query. If there are no records returned, there are no KPIs to compute, and the widgets will display the blue icon that means “no data”.

```
def streamingReproStartFailure =  
avg(streaming_repro_start_failure)  
def streamingServiceNonAcc = avg(streaming_service_non_acc)  
  
def streamingRebuffFailureRatio =  
avg(streaming_rebuff_failure_ratio)  
def streamingReproCutoffRatio =  
avg(streaming_repro_cutoff_ratio)  
  
def streamingRebuffTimePct = avg(streaming_rebuff_time_pct)  
def streamingTimeToStreamStart =  
avg(streaming_time_to_stream_start)
```

For each KPI, the score is equal to the average of all records returned (for the time period)

```
def streamingAcc = 100 - max(0,  
                           degradation(streamingReproStartFailure, 5),  
                           degradation(streamingServiceNonAcc, 5)  
                           )
```

Computes the Accessibility QoE score which is 100%, minus the highest degradation for the two KPIs considered in this category: Reproduction Start Failure Ratio, and Service Non-Accessibility. Each KPI is measured against a threshold of 5%.

Note that increasing the threshold decreases the degradation score, hence results in better QoE scores. The thresholds can be adjusted to better reflect the expected quality of service.

```
def streamingRet = 100 - max(0,
    degradation(streamingRebuffFailureRatio, 5),
    degradation(streamingReproCutoffRatio, 5)
)
```

Computes the Retainability QoE score which is 100%, minus the highest degradation for the two KPIs considered in this category: Rebuffering Failure Ratio, and Reproduction Cut-off Ratio. Each KPI is measured against a threshold of 5%.

```
def streamingQual = 100 - max(0,
    degradation(streamingRebuffTimePct, 25),
    degradation(streamingTimeToStreamStart, 5000)
)
```

Computes the Quality QoE score which is 100%, minus the highest degradation for the two KPIs considered in this category: Rebuffering Time Pct (compared to a threshold of 25%), and Time to Stream Start (compared to a threshold of 5 seconds or 5000 milli-seconds)

```
def streaming = 0.4 * streamingAcc + 0.3 * streamingRet +
0.3 * streamingQual
```

Computes the overall service score, which is a weighted average of the three categories, Accessibility, Retainability and Quality.

```
if (isAdmin) {
    kpi kpiprefix + "streaming", streaming,
    printPercent(streaming)

    kpi kpiprefix + "streaming-acc", streamingAcc,
    printPercent(streamingAcc)
    kpi kpiprefix + "streaming-repro-start-failure-ratio",
    streamingReproStartFailure,
    displayKpiDegradationPct(streamingReproStartFailure, 5)
    kpi kpiprefix + "streaming-service-nonaccessibility",
    streamingServiceNonAcc,
    displayKpiDegradationPct(streamingServiceNonAcc, 5)

    kpi kpiprefix + "streaming-retain", streamingRet,
    printPercent(streamingRet)
    kpi kpiprefix + "streaming-rebuff-failure-
ratio", streamingRebuffFailureRatio,
    displayKpiDegradationPct(streamingRebuffFailureRatio, 5)
    kpi kpiprefix + "streaming-repro-cutoff-ratio",
    streamingReproCutoffRatio,
    displayKpiDegradationPct(streamingReproCutoffRatio, 5)

    kpi kpiprefix + "streaming-qual", streamingQual,
    printPercent(streamingQual)
    kpi kpiprefix + "streaming-rebuff-time-pct",
    streamingRebuffTimePct,
    displayKpiDegradationPct(streamingRebuffTimePct, 25)
    kpi kpiprefix + "streaming-time-to-stream-start",
    streamingTimeToStreamStart,
    displayKpiDegradationSeconds(streamingTimeToStreamStart, 5000)
}
else
```

```

    {
        kpi kpiprefix + "streaming", streaming

        kpi kpiprefix + "streaming-acc", streamingAcc
        kpi kpiprefix + "streaming-repro-start-failure-
ratio", streamingReproStartFailure
        kpi kpiprefix + "streaming-service-
nonaccessibility", streamingServiceNonAcc

        kpi kpiprefix + "streaming-retain", streamingRet
        kpi kpiprefix + "streaming-rebuff-failure-
ratio", streamingRebuffFailureRatio
        kpi kpiprefix + "streaming-repro-cutoff-ratio",
streamingReproCutoffRatio

        kpi kpiprefix + "streaming-qual", streamingQual
        kpi kpiprefix + "streaming-rebuff-time-pct",
streamingRebuffTimePct
        kpi kpiprefix + "streaming-time-to-stream-start",
streamingTimeToStreamStart,
displayKpiDegradationSeconds (streamingTimeToStreamStart, 5000)
    }

```

Finally, assigns the computed values and QoE scores to the associated KPIs to update the widgets. If the user has the admin role, the widgets will also display the percentages.

```
return streaming;
```

Finally, returns the overall score to the parent script.

4.6 Config_filesharing.groovy

This script is similar to the Config_streaming.groovy script described above.

The first part of the script describes the tables and fields for the customer query, the location query and the device query. Then calls the function that computes all the KPIs and overall score for the file sharing indicators

4.7 Config_web_browsing.groovy

This script is similar to the Config_streaming.groovy script described above.

The first part of the script describes the tables and fields for the customer query, the location query and the device query. Then calls the function that computes all the KPIs and overall score for the web browsing indicators.

4.8 Config_email.groovy

This script is similar to the Config_streaming.groovy script described above.

The main difference is that while there are tables "report.gn_pdu_email-subscriber-900-timeline" and "report.gn_pdu_email-cell-900-timeline", but there is no table containing email data per brand_model_os.

As a consequence, device_email_qoe is always returned null.

4.9 Config_network.groovy

This script is also similar to the Config_streaming.groovy script described above, but with a few tweaks.

```
def subscriberNetworkTable = "report.gn_app-subscriber-900-timeline"
def subscriberNetworkDnsTable = "report.gn_pdu_dns-subscriber-900-timeline"
def subscriberNetworkSignalingTable = "report.gn_signaling-subscriber-900-timeline"
```

The network KPIs are spread across three different tables. One for the main network KPIs, one for the DNS metrics, and another one for signaling.

```
// Customer KPIs
def sql_nt_failure_ratio = "[min(100, 100*safediv(1*#{gn_app-subscriber-900-timeline-data_call_access_failure_ratio.count-http-failure-count-2:gn_app-subscriber-900-timeline-data_call_access_failure_ratio.count-http-failure-count-2}, 1*#{gn_app-subscriber-900-timeline-data_call_access_failure_ratio.count-http-session-count-1:gn_app-subscriber-900-timeline-data_call_access_failure_ratio.count-http-session-count-1}, 1))] as failure_ratio"
def sql_nt_tcp_rtt = "[ (abs(#{gn_app-subscriber-900-timeline-common.average-tcp-rtt-client-2:gn_app-subscriber-900-timeline-common.average-tcp-rtt-client-2})+abs(#{gn_app-subscriber-900-timeline-common.average-tcp-rtt-server-3:gn_app-subscriber-900-timeline-common.average-tcp-rtt-server-3}))/1000] as tcp_rtt"
def sql_nt_tcp_retransmission_ratio = "[min(100, 100*safediv(1*#{gn_app-subscriber-900-timeline-tcp_retransmission_ratio.accumulate-tcp-retransmissions-1:gn_app-subscriber-900-timeline-tcp_retransmission_ratio.accumulate-tcp-retransmissions-1}, 1*(#{gn_app-subscriber-900-timeline-tcp_retransmission_ratio.accumulate-bearer-packetsup-2:gn_app-subscriber-900-timeline-tcp_retransmission_ratio.accumulate-bearer-packetsup-2})+#{gn_app-subscriber-900-timeline-tcp_retransmission_ratio.accumulate-bearer-packetsdown-3:gn_app-subscriber-900-timeline-tcp_retransmission_ratio.accumulate-bearer-packetsdown-3}), 1))] as tcp_retransmission_ratio"
def sql_nt_access_time = "[#{gn_app-subscriber-900-timeline-common.average-net-latency-1:gn_app-subscriber-900-timeline-common.average-net-latency-1}+abs(#{gn_app-subscriber-900-timeline-common.average-tcp-rtt-client-2:gn_app-subscriber-900-timeline-common.average-tcp-rtt-client-2})+#{gn_app-subscriber-900-timeline-common.average-tcp-rtt-server-3:gn_app-subscriber-900-timeline-common.average-tcp-rtt-server-3}]/1000] as access_time"

read (db, subscriberNetworkTable, sql_olap_dimension,
      sql_nt_failure_ratio, sql_nt_access_time,
      sql_nt_tcp_retransmission_ratio, sql_nt_tcp_rtt
    )
```

This first query will retrieve all the fields we need from the first table.

```

def sql_nt_resolution_failure_ratio = "[min(100,
100*safediv(1*#{gn_pdu_dns-subscriber-900-timeline-
dns_failure_ratio.count-dns-failure-count-1:gn_pdu_dns-
subscriber-900-timeline-dns_failure_ratio.count-dns-failure-
count-1}, 1*#{gn_pdu_dns-subscriber-900-timeline-
dns_failure_ratio.count-dns-resolution-count-2:gn_pdu_dns-
subscriber-900-timeline-dns_failure_ratio.count-dns-resolution-
count-2}, 1))]" as resolution_failure_ratio"
def sql_nt_resolution_time = "gn_pdu_dns-subscriber-900-
timeline-dns_resolution_time.average-net-latency-1:gn_pdu_dns-
subscriber-900-timeline-dns_resolution_time.average-net-
latency-1"
read (db, subscriberNetworkDnsTable, sql_nt_resolution_time)
resolution_time = getVariable(sql_nt_resolution_time)

read (db, subscriberNetworkDnsTable, sql_olap_dimension,
sql_nt_resolution_failure_ratio)

```

Here, we need to retrieve two KPIs from the DNS table. However, if the first one (*resolution failure ratio*) is a formula, returned “as **resolution_failure_ratio**”, the second one is a simple field “**gn_pdu_dns-subscriber-900-timeline-dns_resolution_time.average-net-latency-1:gn_pdu_dns-subscriber-900-timeline-dns_resolution_time.average-net-latency-1**”.

Because of limitations with the Zstore database, it is not possible to query formulas and fields in the same query. You can query multiple formulas at once, or multiple fields at once, but not a mix of both.

This is the reason why we have to make two queries on the DNS table. The first query is for the field, and the resulting recordset is assigned to variable “resolution_time”. The second query is for the formula.

```

def sql_nt_pdp_failure_ratio = "[min(100,
100*safediv(1*#{gn_signaling-subscriber-900-timeline-
pdp_creation_failure_ratio.count-gtp-activate-pdp-reject-
1:gn_signaling-subscriber-900-timeline-
pdp_creation_failure_ratio.count-gtp-activate-pdp-reject-
1}, 1*#{gn_signaling-subscriber-900-timeline-
pdp_creation_failure_ratio.count-gtp-activate-pdp-request-
2:gn_signaling-subscriber-900-timeline-
pdp_creation_failure_ratio.count-gtp-activate-pdp-request-2},
1))]" as pdp_failure_ratio"
def sql_nt_pdp_creation_time = "gn_signaling-subscriber-900-
timeline-pdp_creation_time.average-gtp-latency-1:gn_signaling-
subscriber-900-timeline-pdp_creation_time.average-gtp-latency-
1"
def sql_nt_pdp_cutoff_failure_ratio = "[min(100,
100*safediv(1*#{gn_signaling-subscriber-900-timeline-
pdp_cutoff_ratio.count-gtp-deactivate-pdp-downlink-
1:gn_signaling-subscriber-900-timeline-pdp_cutoff_ratio.count-
gtp-deactivate-pdp-downlink-1}, 1*#{gn_signaling-subscriber-900-
timeline-pdp_cutoff_ratio.count-gtp-deactivate-pdp-all-
2:gn_signaling-subscriber-900-timeline-pdp_cutoff_ratio.count-
gtp-deactivate-pdp-all-2}, 1))]" as pdp_cutoff_failure_ratio"
def sql_nt_rat_type = "[1*#{gn_signaling-subscriber-900-
timeline-rat_type_distribution.count-gtp-rat-utran-
1:gn_signaling-subscriber-900-timeline-
rat_type_distribution.count-gtp-rat-utran-1}] as ratype_utran,
[1*#{gn_signaling-subscriber-900-timeline-
rat_type_distribution.count-gtp-rat-geran-2:gn_signaling-
subscriber-900-timeline-rat_type_distribution.count-gtp-rat-

```

```

geran-2}} as ratype_geran, [1*#{gn_signaling-subscriber-900-
timeline-rat_type_distribution.count-gtp-rat-wlan-
3:gn_signaling-subscriber-900-timeline-
rat_type_distribution.count-gtp-rat-wlan-3}} as ratype_wlan,
[1*#{gn_signaling-subscriber-900-timeline-
rat_type_distribution.count-gtp-rat-gan-4:gn_signaling-
subscriber-900-timeline-rat_type_distribution.count-gtp-rat-
gan-4}} as ratype_gan, [1*#{gn_signaling-subscriber-900-
timeline-rat_type_distribution.count-gtp-rat-hspa-
5:gn_signaling-subscriber-900-timeline-
rat_type_distribution.count-gtp-rat-hspa-5}} as ratype_hspa"
read (db, subscriberNetworkSignalingTable,
sql_nt_pdp_creation_time)
pdp_creation_time = getVariable(sql_nt_pdp_creation_time);
read (db, subscriberNetworkSignalingTable, sql_olap_dimension,
sql_nt_pdp_failure_ratio,
sql_nt_pdp_cutoff_failure_ratio, sql_nt_rat_type)

```

Again, we need to make two separate queries on the Signaling table, to get back a set of formulas, and a single field is a separate query.

```
customer_network_qoe = buildNetworkKpis("customer-");
```

The script has executed 5 queries to the Zstore, which have populated a number of variables. These variables will be used in the “buildNetworkKPIs” function to compute the KPIs for the network.

We then have the equivalent queries executed on the “per cell” tables, and “per device” tables.

4.10 Icons

The MBBQoE customization mostly uses icons taken from the HP Experience Live library, which are packaged with the CCD kit.

This icons can be referenced via the path “resources/HP_exp-live”:

```
icon="resources/HP_exp-
live/img/icons/file_sharing_48.png"
```

The MBBQoE customization also provides two custom icons, to demonstrate how the view can be customized. These icons are packaged in the folder /resources/icons, and can be referenced in the configGUIView.xml through this path: “resources/mbbqoe/”:

```
icon="resources/mbbqoe/File_Sharing_sharing_RGB_gray_48.p
ng"
```



4.11 Messages

All the labels in the MBBQoE customization can be easily modified, and localized.

/resources/properties/MessagesBundle.properties contains a set of messages, in the form of key-value pairs.

These messages can be used from configGUIView.xml, or from the Config.groovy script.

In configGUIView.xml, each widget can have a “title” attribute. This attribute is matched into the ResourceBundle properties file, and if found, the title of the widget is replaced by the text property. Otherwise, the content of the title attribute will be used as the widget title.

In Config.groovy (or any other groovy), it is possible to use the “l10n” keyword to search for any label in the messages file.

For instance:

```
String displayKpiDegradationSeconds(value, threshold)
{
    if (value == null) return "";
    def deg = degradation(value, threshold);
    return concat (printSeconds (value),
        " (", l10n("threshold"), ":",
printSeconds (threshold), ") ",
        l10n("degradation"), ":", printPercent (deg),
" ",
        l10n("QoE"), ":", printPercent (100 - deg)
    )
}
```

In addition to providing an easy way to isolate and update labels, the ResourceBundle mechanism allows to easily localize the client’s view.

If the user accesses the dashboard with a “locale” parameter, CCD will try to find a properties file named ResourceBundle_<locale>.properties.

If this file exists, then it will get the labels from this localized file.

We provide in the customization kit an example of a French translation of the custom.