



# FABRIC 6.1

## Runtime User Guide

April, 2020

## Confidentiality

This document contains copyrighted work and proprietary information belonging to K2View.

This document and information contained herein are delivered to you as is, and K2View makes no warranty whatsoever as to its accuracy, completeness, fitness for a particular purpose, or use. Any use of the documentation and/or the information contained herein, is at the user's risk, and K2View is not responsible for any direct, indirect, special, incidental, or consequential damages arising out of such use of the documentation. Technical or other inaccuracies, as well as typographical errors, may occur in this Guide.

This document and the information contained herein and any part thereof are confidential and proprietary to K2View. All intellectual property rights (including, without limitation, copyrights, trade secrets, trademarks, etc.) evidenced by or embodied in and/or attached, connected, or related to this Guide, as well as any information contained herein, are and shall be owned solely by K2View. K2View does not convey to you an interest in or to this Guide, to information contained herein, or to its intellectual property rights, but only a personal, limited, fully revocable right to use the Guide solely for reviewing purposes. Unless explicitly set forth otherwise, you may not reproduce by any means any document and/or copyright contained herein.

Information in this Guide is subject to change without notice. Corporate and individual names and data used in examples herein are fictitious unless otherwise noted.

Copyright © 2020 K2View Ltd./K2VIEW LLC. All rights reserved.

The following are trademark of K2View:

K2View logo, K2View's platform.

K2View reserves the right to update this list from time to time.

Other company and brand products and service names in this Guide are trademarks or registered trademarks of their respective holders.

# Contents

<b>1</b>	<b>Introduction to K2View Fabric Database .....</b>	<b>6</b>
<b>2</b>	<b>Concepts and Terminology .....</b>	<b>7</b>
2.1	Abbreviations.....	7
2.2	Main Concepts .....	7
<b>3</b>	<b>Getting Started .....</b>	<b>8</b>
3.1	Configuration.....	8
3.2	Main Keyspaces .....	9
3.2.1	k2system .....	9
3.2.2	K2auth.....	9
3.2.3	k2batchprocess.....	10
3.2.4	K2audit.....	10
3.3	Main Directories .....	10
3.3.1	Temp Files Directory .....	10
3.4	Fabric Login .....	10
<b>4</b>	<b>K2View Fabric Basic Commands .....</b>	<b>11</b>
4.1	Get Command.....	11
4.1.1	Support get@DC getf@DC .....	11
4.2	Set Commands.....	12
4.2.1	Set Sync .....	12
4.2.2	Set Variable on Session .....	13
4.2.3	Set Output.....	13
4.2.4	Set .....	14
4.2.5	Set Default .....	14
4.2.6	Set Environment .....	14
4.2.7	Set Global Environment.....	14
4.2.8	Set Global Global .....	14
4.2.9	Set Passforward .....	15
4.2.10	Set instance_ttl .....	15
4.2.11	Set sync_timeout.....	15
4.2.12	Set ignore_source_exception.....	15
4.3	Offline Deploy .....	16
4.4	Run Batch Processes in Fabric .....	17
4.4.1	Batch Process- Fabric 6.0 Enhancements .....	17
4.4.2	Run Batch Process Commands .....	18
4.5	Migrate Instances into Fabric Database.....	22
4.6	Batch Processing Monitoring Commands.....	26
4.7	Migrate Monitoring Commands .....	28
4.8	Resume (Retry) Batch Process .....	30
4.9	Resume Migrate.....	31
4.9.1	General.....	31

4.10	Cancel Batch Command .....	32
4.10.1	General .....	32
4.11	Security Definitions .....	32
4.11.1	General .....	32
4.11.2	K2auth Tables .....	32
4.11.3	Get Users Information from Cassandra .....	33
4.11.4	Add Users .....	33
4.11.5	Add Roles & Token .....	34
4.11.6	Drop .....	34
4.11.7	Grant Command .....	34
4.11.7.1	Grant All .....	35
4.11.7.2	Grant Web Service (Method) .....	35
4.11.7.3	Grant Operation .....	36
4.11.8	Revoke Command .....	36
4.11.9	Check Permissions .....	36
4.12	Describe Command .....	37
4.13	List Command .....	38
4.14	Delete Instances Commands .....	40
4.15	Queries Helpers .....	40
4.16	Search/CDC .....	41
4.17	Broadway .....	41
4.18	Other .....	42
4.19	Reference Tables Commands .....	42
4.20	Parser Commands .....	43
4.20.1	Add a New Configuration File- node.id .....	46
4.20.2	Parser Command .....	46
4.21	Job Mechanism Commands .....	48
4.21.1	Job Commands .....	49
4.21.2	Fabric V3.6- Cassandra Loader (Bulk Loader) .....	50
4.22	Cloning Fabric Session .....	52
4.23	URL Rewriting .....	52
4.24	Salesforce Support .....	53
<b>5</b>	<b>Fabric as System of Record .....</b>	<b>54</b>
5.1	Update LU Instance - Synchronic Mode .....	54
5.1.1	Synchronic Update of LUI- Examples .....	56
5.1.1.1	Implementation code - run transaction using fabric user code methods: .....	56
5.1.1.2	Run Transaction Using the Fabric Console .....	56
5.2	Update LU Instance-Asynchronous Mode .....	57
5.3	Update Common Tables .....	57
<b>6</b>	<b>Fabric API Documentation .....</b>	<b>59</b>
<b>7</b>	<b>Log Tracking .....</b>	<b>60</b>
7.1	Configuration File .....	60
7.2	Log File .....	61

7.2.1	Add Connection Leaks to the Log File .....	61
7.3	Log Table .....	61
7.3.1	Table Structure .....	61
7.3.2	Error Code .....	62
8	<b>Servlet Filters .....</b>	<b>63</b>
9	<b>Enable GZIP Compression of Web Service Response .....</b>	<b>64</b>
10	<b>Trace Mechanism to Fabric .....</b>	<b>65</b>
11	<b>JMX - Java Management Extensions.....</b>	<b>67</b>
11.1	Custom JMX Metrics .....	67
12	<b>Fabric Auditing.....</b>	<b>69</b>
12.1	Custom Auditing.....	70
12.1.1	Build Custom Audit.....	70
12.1.2	Filter Activities for Audit .....	70
12.1.3	Custom Persistency Strategy .....	72
13	<b>Cleanup Procedures.....</b>	<b>73</b>
13.1	Drop All LUs (keyspace) .....	73
13.2	Drop Specific LU (keyspace).....	73
13.3	Switching Between the Global Environments .....	73
13.4	Switching Between the Session Environments.....	74
13.5	Checking Active Environment .....	75
13.6	Deploy Enviroments (From File).....	75
13.7	Setting Globals .....	76
13.8	Permissions .....	76
14	<b>Web Admin .....</b>	<b>77</b>
14.1	Managing Permissions.....	77
14.2	Managing Roles .....	79
14.3	Managing Tokens .....	80
14.4	Managing Users.....	82
14.5	Viewing Web Services .....	84
14.6	Displaying Data .....	85
14.7	Running Batch Processes.....	87
14.8	Viewing the Batch Process Executions .....	88
15	<b>Customer Support.....</b>	<b>90</b>

---

# 1 Introduction to K2View Fabric Database

K2View Fabric is an innovative, revolutionary, real-time data layer residing on top of Apache Cassandra. It provides an easy, secure and reliable way to consolidate your data and distribute it over your network for high availability. For more information, see the *Introduction to K2View Fabric* white paper.

K2View Fabric consists of two distinct layers:

- Implementation Layer: the design, development, definition of tables and transformation rules and deployment are performed in this environment.
- Execution Layer: the actual migration and synchronization of data from the source systems to K2View Fabric is performed in this environment.

This document describes the K2View Fabric execution layer.

---

## 2 Concepts and Terminology

---

### 2.1 Abbreviations

- LU: Logical Unit
- LUT: Logical Unit Type
- LUI: Logical Unit Instance
- ERD: Entity Relationship Diagram

---

### 2.2 Main Concepts

This chapter describes the main concepts of K2View Fabric:

- Logical Unit Type/Schema: The Logical Unit is one of K2View Fabric' foundations. The Logical Unit is a business view of your data. This is a relational schema of tables.
- Table: An object which contains data either directly from one or more source tables or as calculated data.
- Root Table: The primary table of the LU's ERD.
- Instance KEY: A unique field that will be used as the logical unit instance main identifier. A logical unit type (LUT) can contain many instance KEYs. For example: customer id 12345, which represents a single customer is an instance KEY in the CUSTOMER logical unit type (schema).
- Instance Group: A list of instance KEYs to be processed.
- Sync: K2View Fabric features an intelligent and flexible way to synchronize data: AlwaysSync. This mode allows complete granularity over the data that should be synchronized with source systems. Using AlwaysSync, K2View Fabric supports configuration of what data needs to be refreshed automatically, and how frequently. For each table population within the Schema, an AlwaysSync timer is set to drive the K2View Fabric synchronization (for example, if the usage information table population from the Customer Schema needs to be updated every 5 minutes, a timer of 5 minutes is set). Once configured, K2View Fabric will synchronize data with source systems only on data access to optimize performances by avoiding overload of the system for synchronization of unused data. As such, when retrieved, if the timestamp on the logical unit instance data is older than the timer set for this data, K2View Fabric triggers synchronization.

## 3 Getting Started

K2View Fabric uses Cassandra DB as a storage layer. You can, therefore, connect to it using any driver that connects to Cassandra.

### CQL reference:

[http://www.datastax.com/documentation/cql/3.1/cql/cql\\_intro\\_c.html](http://www.datastax.com/documentation/cql/3.1/cql/cql_intro_c.html)

Client list: <http://planetcassandra.org/client-drivers-tools/>

It is required to install Cassandra Vanilla, then link Fabric (in config.ini) to the installed Cassandra release, and bring Cassandra up.

Through command line:

- To start K2View Fabric run, type: `k2fabric start`

**Note:** When starting Fabric, you will receive notifications if you have some local files that are in conflict with the installed release (private files).

- To get the installed Fabric version on your server, type: `k2fabric -version`
- To stop K2View Fabric run, type: `k2fabric stop`
- To enter K2View Fabric run, type: `fabric`
- To enter Cassandra run, type: `./cqlsh -u <user> -p <password> <ip address>`

### 3.1 Configuration

After installing the K2View Fabric according to the K2View Fabric Installation Guide, configure K2View Fabric parameters within the configuration file.

- File location - `/usr/local/k2view/config/config.ini`
- Sample parameters:
  - ◆ `JOB_SERVER_WORKERS_COUNT=8`  
Number of workers
  - ◆ `MDB_DEFAULT_SCHEMA_CACHE_SIZE=10000000`
  - ◆ Cache size limit in bytes per Schema for MicroDB instances that are not currently in use.



## 3.2 Main Keyspaces

### 3.2.1k2system

The table below details the key space tables:

Table Name	Description
K2_lut_info	This table holds definitions of LUs, Reference and Web Services deployed through the K2View Fabric Studio.
Log	Will be described in the log section (5).
k2_jobs	This table contains information on the history of all the jobs run on Fabric.
nodes	List of all nodes in the cluster.
tasks	List of tasks used by the scheduler.
global_settings	List of all GLOBALS/Environments that were overridden from the default value in order to identify what is the override value when restarting Fabric and etc.
node_batches_info	This table contains the statistics per node for a batch id when running MPP queries.
lut_action_history	This table contains the previous activity run on a given LUT with timestamp (for example: if LUT 'PATIENT_LU' was deployed, dropped and deployed again, this table will show when 'PATIENT_LU' was dropped).

### 3.2.2K2auth

This keyspace holds K2View Fabric security definitions for accessing K2View Fabric data through console or web services.

Table Name	Description
user_credentials	List of users with the assigned roles.
roles	List of role definitions.
credentials	List of tokens with assigned roles.
permissions	List of permissions for a given role.
stripe_key_storage	Contains the stripped master key encrypted info.
keys_descriptions	Contains master key descriptions.

### 3.2.3k2batchprocess

This keyspace holds K2view Fabric batch processes, including migration, information.

Table Name	Description
batch_list	List of the entire history of the batch process commands.
batch_node_info	Summary of handled entities per batch process per node.
batch_entities_info	Detailed execution information for a given entity per batch process command.
batch_entities_errors	Detailed entities information for failed entities (instances) per migration execution. This table is intended to improve the search performance for failed instances in migration.

### 3.2.4K2audit

Table Name	Description
K2_auditing	Every activity performed within K2View Fabric is logged into this table.

## 3.3 Main Directories

### 3.3.1Temp Files Directory

This directory holds the cached database files. The default directory is `/dev/shm/fdb_cache/<LU NAME>/` (as defined in the K2View Fabric config file).

Example: `6.db`

**Note:** 6 is incremental sequence

## 3.4 Fabric Login

In order to login into Fabric console, run Fabric script.

## 4 K2View Fabric Basic Commands

This chapter will guide and teach you how to access Fabric data through web services, Fabric viewer tools or Fabric console.

You can view Fabric table data, run SQL statements, set user permissions for web services, migrate instances into Fabric etc.

After you log in to Fabric, type **Help** to view the list of available Fabric commands.

To view the syntax of a specific command, type **help <command>**. For example, **help grant**.

The following sections describe the most common commands.

### 4.1 Get Command

The GET command is used to bring information for a given schema in Fabric and synchronize information from the source systems, only if needed (see the Sync term in the Concepts and Terminology section).

Command Name	Description	Syntax	Example
Get Specific Instance	This command is used to bring information for a specific instance. If the instance does not exist in Fabric, it will be migrated from the source systems into Fabric in real time.	get <LUT>.<Instance ID>;	get CUST.400000001;  Customer data for ID 400000001 is fetched from the source systems into CUST keyspace within Fabric
GetF- get instance id by function	This command is used to bring information for a specific instance that populate from function with input parameters	getF <LUT>.<function name>(829767918 1);	getF P7.func_oneInp(8297679181)  Customer data 8297679181 is fetched from Fabric into CUST keyspace within fabric

#### 4.1.1Support get@DC getf@DC

This new feature is required when you execute a get command from a DC which is not connected to any sources, while other DCs are connected to the source interfaces. The new get@DC or getf@DC commands invoke the remote DC (connected to the source). Invoking the command on the remote DC will be done via JDBC. The remote GET/GETF command will return after the GET/GETF command has finished execution on the remote node. Cassandra then, replicates the data between the nodes of the cluster.

- Syntax
  - ◆ `get <LUT Name>.<Instance id>@<DC name>;`
  - ◆ `getf <LUT_Name>.<function name>(arg... )@<DC name>;`

**Notes:** It is the user's responsibility to identify if sync is required, and only then run the remote GET/GETF commands. This will prevent unnecessary calls to the remote node, so the file will be fetched locally.  
The node that will run the GET/GETF on the remote DC will be selected randomly.  
Verifying permissions for the GET/GETF execution will be done on both nodes - local and remote.

## 4.2 Set Commands

### 4.2.1 Set Sync

Sync command is used in order to define the synchronization from the source systems as explained in the above Terminology section (2).

The default value is ON.

Syntax: `set sync <Sync Mode>`

Synchronization is performed in the following scenarios:

Sync Mode	Single Instance
ON	<p>Schema was changed and redeployed.</p> <p>Instance does not exist in Fabric yet.</p> <p>According to the pre-defined Sync interval.</p> <p>Sync will return error message, if source is not available. In order to change this behavior, use <code>set IGNORE_SOURCE_EXCEPTION=true</code> command.</p>
OFF	<p>Synchronization will not be performed.</p> <p>In case Instance does not exist in Fabric database yet, a warning message is displayed.</p>
FORCE	<p>Synchronization is performed for every operation on Fabric database instance, regardless of the sync interval definitions in the K2View Fabric Studio.</p>

### 4.2.2 Set Variable on Session

- Description:  
This set command can be used in order to set variable on the session to be used later on by the implemented code.
- Syntax:
- SET KEY=VALUE; or SET KEY VALUE;
- SET KEY="; - will remove the variable from session
- SET KEY; - in order to get the KEY VALUE;
- SET instance\_ttl <TTL>; | Set time to live (TTL) for each entity (LUI)

### 4.2.3 Set Output

- Description:  
This set command can be used in order to save query results into an output file. After running it, each query result will be appended to the file until set output is turned off.
- Syntax:  
Set output file '<full path to file >' [with (options...)]  
Where options can be one of the following:

```
FORMAT: [text|csv]
DELIMITER: 'delimiter_character'
NULL: 'null_string'
HEADER: [true|false]
QUOTE: 'quote_character'
ESCAPE: 'escape_character'
FORCE_QUOTE: [true|false]
ENCODING : 'encoding_name'
APPEND: [true|false]
```

- To turn the feature off, write 'set output stdout;'
- File location:

If full path to file is not provided, the file will be created under /usr/local/k2view/export

- Example:
  - ◆ Set output file "mytestcsv.csv" with header=true and delimiter="\*" and append=false;
  - ◆ Set output file "mytestcsv.csv" with header=false and delimiter="\*" and append=false and null="kuku";
  - ◆ Set output file "/usr/local/k2view/fabric/files/mytestcsv3.csv" with header=false and delimiter="\*" and append=true and null="kuku";

#### 4.2.4 Set

- Description:  
This command displays the current session's values for the above-mentioned parameters, as well as the session's scope and globals.
- Syntax:
  - ◆ set

#### 4.2.5 Set Default

- Description:  
This command resets all the set-related parameters to default value.
- Syntax:
  - ◆ Set default

#### 4.2.6 Set Environment

- Description:  
This command sets the environment for the current session.
- Syntax:
  - ◆ set environment='<environment name>'
- Example:

```
set environment='env1' - set the environment to env1 settings
```

```
set environment="" - reset the session environment
```

#### 4.2.7 Set Global Environment

- Description:  
This command sets the environment for the whole cluster.
- Syntax:
  - ◆ Set\_global environment='<environment name>'

#### 4.2.8 Set Global Global

- Description:  
This command overrides global value that is not defined as final on the whole cluster.  
  
Syntax for a given LUT:  
set\_global global '<LUTNAME>.<PARAM\_NAME>[=<PARAM\_VALUE>]'
- Syntax for all LUTs:

- ◆ `set_global global '.*.<PARAM_NAME>[=<PARAM_VALUE>]'`

#### 4.2.9 Set Passforward

- Description:  
When parsing a cql statement, the command will define if the statement will be passed to SQLite, if an error is found in the statement's pre-parsing, or not.
- Syntax:  
`Set passforward <on|off>`

#### 4.2.10 Set instance\_ttl

- Description:
  - ◆ Set time to live (TTL) for each entity (LUI)
  - ◆ TTL - is set in seconds in case of LUI sync from source.
- Syntax:
  - ◆ `Set instance_ttl <TTL>`

#### 4.2.11 Set sync\_timeout

- Description:
  - ◆ Set sync timeout in seconds
- Syntax:
  - ◆ `set sync_timeout=<sync_timeout_sec>`
  - ◆ `set sync_timeout=''` - will set timeout back to LU default value

#### 4.2.12 Set ignore\_source\_exception

- Description:
  - ◆ Sets the error handling when Fabric fails to connect the source to sync the entity (instance). The default value is false.
- Syntax:
  - ◆ `set ignore_source_exception <true/false>`
  - ◆ true: if Fabric fails to connect to source to sync the instance, no exception is given, but the changes are rollbacked, and the instance version is not updated.
  - ◆ false: an exception is thrown if Fabric fails to connect the source to sync the instance.

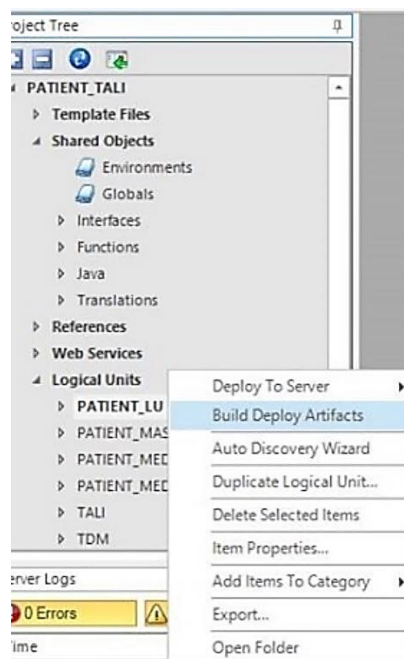
## 4.3 Offline Deploy

- Description:

The Deploy command can be performed from Fabric server using artifices that are created by Fabric Studio.

The following steps should be performed:

- ◆ Create studio artifact by right-clicking the LU and selecting 'Build Deploy Artifacts'.



- ◆ Once the artifacts are created, remove the following items from file: ludb.jar, ludb.json and ludbXMLs.zip (it is recommended to save the artifacts in the SVN)
- ◆ Move the artifacts to the server
- ◆ Run the deploy command using the following syntax.

Syntax:

```
DEPLOY <LUT> WITH JAR <'jar_path'> ZIP_FILE <'zip path'>
[WS_METHODS <'string'>] NOSYNC <boolean>
```

Example:

```
DEPLOY parswithNull WITH JSON '${ lu_path}/ludb.json' JAR
'${ lu_path}/ludb.jar' zip_file '${ lu_path}/ludbXMLs.zip'
NOSYNC TRUE;
```



**Note:** If the LUT parameter is populated by 'k2\_ws' (web-service lu type), you can populate the WS\_METHODS by the list of web services to be deployed. If this parameter is not populated (or empty), the deploy command deploys all the WS into the fabric.

- From release 4.0 it is possible to build the artifacts on server side. Two new scripts are located under /usr/local/k2view/fabric/scripts:
  - ◆ buildArtifacts.sh for Linux or buildArtifacts.bat for windows
    - Description: Build only Artifact files
    - Usage: ./buildArtifacts.sh <PATH\_TO\_PROJECT> <LUTNAME>
    - If LU type name is not provided, it generates the artifacts for all the LUs
  - ◆ buildAndDeployArtifacts.sh for Linux or buildAndDeployArtifacts.bat for Windows
    - Description: Deploy and build artifact files
    - Usage: ./buildAndDeployArtifacts.sh <PATH\_TO\_PROJECT> <NOSYNC> [LUTNAME] [USER] [PASSWORD] [DEPLOYONLY]
    - LUTNAME is mandatory

## 4.4 Run Batch Processes in Fabric

Batch process commands allow executing different types of fabric commands on the remote Fabric nodes. Currently, the following batch processes are supported:

- ◆ Migrate instances into Fabric
- ◆ Execute Broadway workflows on Fabric

### 4.4.1 Batch Process- Fabric 6.0 Enhancements

- ◆ Enable adding new nodes to Fabric cluster during the execution of the batch process. The batch process allocates instances to the new nodes.
- ◆ Replace k2migrate key space by k2batchprocess key space and replace the migrate Cassandra tables by batch process Cassandra tables.
- ◆ Add new batch process commands.

**Note:** You can use both new batch process commands and current migrate commands to run or monitor the migrate process. The migrate commands remain for compatibility mode and activate the batch process commands behind the scenes.

- ◆ Replace JOB\_SERVER\_WORKERS\_COUNT parameter by MAX\_WORKERS\_PER\_NODE parameter to set the maximum number

of threads that are used in all batch process units (executions) on this node.

- ◆ Support for MAX\_WORKERS\_PER\_NODE=0 per node to avoid running a batch process on a specific node.
- ◆ Create one k2jobs record (type = 'BATCH\_JOB') per migrate command. Each execution creates a new record in k2batchprocess.batchprocess\_list Cassandra table.
- ◆ New JMX metrics:
  - batchProcessCompletedSuccessfullyCounter
  - batchProcessFailedCounter
  - batchProcessFatalFailedCounter

#### 4.4.2Run Batch Process Commands

Command Name	Description	Syntax	Example
batch	Start a batch process on all instances of a Logical Unit.	<p>batch &lt;LUT&gt;[@&lt;DC&gt;]            FABRIC_COMMAND='&lt;fabric command&gt; ?' [WITH            [AFFINITY='&lt;affinity&gt;']            [JOB_AFFINITY='&lt;job affinity&gt;']            [ASYNC=true/false]            [GENERATE_ENTITIES_FIR            T=true/false]            [ALLOW_MULTIPLY=true/f            alse]            [MAX_WORKERS_PER_NOD            E=&lt;number&gt;]];</p> <p>DC - specify DC name to force the batch process within specified DC. It can also be defined on the affinity parameter, remained for backward compatibility.</p> <p>AFFINITY - list of nodes and DCs to be involved in the batch command .</p> <p>JOB_AFFINITY - affinity for the batch process job.</p> <p>ASYNC - defines if batch process should run on sync or async mode, default is false.</p>	<p>batch PATIENT            FABRIC_COMMAND="sync_in            stance PATIENT.?" with            async='true';</p> <p>This command migrates all patients from the source systems into Fabric PATIENT keyspace within the Fabric database.</p>

Command Name	Description	Syntax	Example
		<p>GENERATE_ENTITIES_FIRST - if set to true - generates all entities before starting to process them.</p> <p>FABRIC_COMMAND - Fabric command to be executed by the batch process. It can be any command that includes one '?' that will represent entity id. For migration, the following command must be set: "sync_instance &lt;LUT&gt;?.?"</p> <p>ALLOW_MULTIPLY - set to 'true' to allow multiple executions of the same batch process command (default is false).</p> <p>MAX_NODES - the maximum nodes that will participate in the batch process (randomly nodes).</p> <p>MAX_WORKERS_PER_NODE - enables setting a smaller number of maximum workers to run on each node, than the maximum number of workers, defined in the config.ini file (MAX_WORKERS_PER_NODE parameter).</p>	
Run the batch process on a defined List of Instances (instance group)	Use this command in order to run the batch process on a selective list of instances. The list should be predefined in the K2View Fabric Studio as explained in K2View Fabric Studio user guide under the 'Instance Group' section.	<p>batch &lt;LUT&gt;[@&lt;DC&gt;].&lt;IG&gt;  fabric_command='&lt;fabric command&gt; ?' [WITH  [AFFINITY='&lt;affinity&gt;']  [JOB_AFFINITY='&lt;job affinity&gt;']  [ASYNC=true/false]  [GENERATE_ENTITIES_FIRST=true/false]  [ALLOW_MULTIPLY=true/false]  [MAX_NODES=&lt;number&gt;]</p>	<p>batch CUST.  ig10CustomersList  FABRIC_COMMAND="sync_instance CUST.?" with  async='true';</p> <p>This command migrates the customers that are defined in 'ig10CustomersList' into CUST schema in Fabric.</p>

Command Name	Description	Syntax	Example
		[MAX_WORKERS_PER_NODE=<number>];	
Run the batch process on a list of instances based on the SQL input parameter.	Use this command to run the batch process on a selective list of instances from a source database into the Fabric database. The list is built based on the input SQL.	batch <LUT>[@<DC>] from <db_interface> using ('<SQL>') fabric_command='<fabric command> ?' [WITH [AFFINITY='<affinity>'] [JOB_AFFINITY='<job affinity>'] [ASYNC=true/false] [GENERATE_ENTITIES_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=<number>] [MAX_WORKERS_PER_NODE=<number>]];	batch PATIENT FROM HIS_DB USING ('select patient_id from PATIENT where patient_id <= 1000') FABRIC_COMMAND="sync_instance PATIENT.?" with async='true';
Run the batch process on a subset of Logical Unit Instances, based on list of instances.	Use this command to run the batch process on a selective list of instances provided as input to the command.	batch <LUT>[@<DC>].(<instance 1,instance 2,etc...>) fabric_command='<fabric command> ?' [WITH [AFFINITY='<affinity>'] [JOB_AFFINITY='<job affinity>'] [ASYNC=true/false] [GENERATE_ENTITIES_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=<number>] [MAX_WORKERS_PER_NODE=<number>]];	batch PATIENT.('dev_1', 'dev_2', 'dev_3') FABRIC_COMMAND="sync_instance PATIENT.?" with async='true';
batchf - run the batch process on a list of instances by function.		1) batchf <LUT>[@<DC>].<function>() FABRIC_COMMAND='<fabric command> ?' [WITH [AFFINITY='<affinity>'] [JOB_AFFINITY='<job affinity>'] [ASYNC=true/false] [GENERATE_ENTITIES_FIRST=true/false]	See above.

Command Name	Description	Syntax	Example
		<pre>[ALLOW_MULTIPLY=true/false] [MAX_WORKERS_PER_NODE=&lt;number&gt;]];  2) batchf &lt;LUT&gt;[@&lt;DC&gt;].&lt;function&gt;().&lt;IG&gt; fabric_command='&lt;fabric command&gt; ?' [WITH [AFFINITY='&lt;affinity&gt;'] [JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_ENTITIES_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=&lt;number&gt;] [MAX_WORKERS_PER_NODE=&lt;number&gt;]];  3) batchf &lt;LUT&gt;[@&lt;DC&gt;].&lt;function&gt;() from &lt;db_interface&gt; using ('&lt;SQL&gt;') fabric_command='&lt;fabric command&gt; ?' [WITH [AFFINITY='&lt;affinity&gt;'] [JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_ENTITIES_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=&lt;number&gt;] [MAX_WORKERS_PER_NODE=&lt;number&gt;]];  4) batchf &lt;LUT&gt;[@&lt;DC&gt;].&lt;function&gt;().(&lt;instance 1,instance 2,etc...&gt;) fabric_command='&lt;fabric command&gt; ?' [WITH [AFFINITY='&lt;affinity&gt;']</pre>	

Command Name	Description	Syntax	Example
		<pre>[JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_ENTITIES_ FIRST=true/false] [ALLOW_MULTIPLY=tr ue/false] [MAX_NODES=&lt;number &gt;] [MAX_WORKERS_PER_ NODE=&lt;number&gt;]];</pre>	

## 4.5 Migrate Instances into Fabric Database

Fabric also supports the migrate commands for compatibility mode. Behind the scenes, Fabric activates the batch command when running migrate commands.

Command Name	Description	Syntax	Example
Migrate All Instances	Use this command in order to migrate instances from the source systems into Fabric database for a given LU Type.	<pre>migrate &lt;LUT&gt;[@&lt;DC&gt;] [WITH [AFFINITY='&lt;affinity&gt;'] [JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_IIDS_FIRST=tru e/false] [ALLOW_MULTIPLY=true/f alse] [MAX_NODES=&lt;number&gt;] [MAX_WORKERS_PER_NOD E=&lt;number&gt;]];</pre>	<pre>migrate CUST;</pre> <p>This command migrates all customers from the source systems into Fabric CUST keyspace within the Fabric database.</p> <p>DC - optional to run the migrate on local DC. It is possible to use the AFFINITY instead, remained due to backwards compatibility.</p> <p>AFFINITY is an optional parameter for choosing a list of nodes and DCs to be involved in the migrate command.</p> <p>JOB_AFFINITY - Affinity for the migrate job (coordinator node).</p>

Command Name	Description	Syntax	Example
			<p>ASYNC - is an optional parameter that overrides everything that was defined on the session.</p> <p>GENERATE_IIDS_FIRST - if set to true, generates all iids before starting to process them. Default value is false.</p> <p>ALLOW_MULTIPLY - set to 'true' to allow multiply execution of the same migrate command (default is false).</p> <p>MAX_NODES - the maximum nodes that will participate in the migrate (random nodes).</p> <p>MAX_WORKERS_PER_NODE - enables setting a smaller number of maximum workers to run on each node, than the maximum number of workers, defined in the config.ini file (MAX_WORKERS_PER_NODE parameter).</p>
Migrate a Pre-Defined List of Instances (instance group).	Use this command to migrate a selective list of instances from source database into Fabric database for a given LU Type. The list should be predefined within the K2View Fabric Studio as explained in K2View Fabric Studio user guide under the 'Instance Group' section.	<pre>migrate &lt;LUT&gt;[@&lt;DC&gt;]..&lt;Instance Group Name&gt; [WITH [AFFINITY='&lt;affinity&gt;'] [JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_IIDS_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=&lt;number&gt;] [MAX_WORKERS_PER_NODE=&lt;number&gt;]];</pre>	<pre>migrate CUSTOMER.ig10CustomersList</pre> <p>This command migrates the customers that are defined in 'ig10CustomersList' into CUST schema in Fabric.</p>
Migrate a list of instances based	Use this command to migrate a selective list of instances from a source	<pre>migrate MS[@&lt;DC&gt;]. from &lt; Database Interface&gt; using ('&lt;SQL&gt;') [WITH</pre>	<pre>migrate MS from MYSQL_SOURCE using</pre>

Command Name	Description	Syntax	Example
on the SQL input parameter.	database into Fabric database. The list is built based on the input SQL.	<pre>[AFFINITY='&lt;affinity&gt;'] [JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_IIDS_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=&lt;number&gt;] [MAX_WORKERS_PER_NODE=&lt;number&gt;]];</pre> <p>Use from fabric in order to run a remigration of the existing instances in entity table</p>	('select customer_id from yp_customer limit 100');
Migrate a subset of Logical Unit Instances, based on list of instances.	Use this command to migrate a selective list of instances provided as input to the command.	<pre>migrate &lt;LUT&gt;[@&lt;DC&gt;].(&lt;instance 1,instance 2,etc...&gt;) [WITH [AFFINITY='&lt;affinity&gt;'] [JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_IIDS_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=&lt;number&gt;] [MAX_WORKERS_PER_NODE=&lt;number&gt;]];</pre>	Migrate MS.('1','2','3');
MigrateF - migrate a list of instances by function.	Use this command to migrate a selective list of instances by function.	<pre>1) migratef &lt;LUT&gt;[@&lt;DC&gt;].&lt;function&gt;().&lt;IG&gt; [WITH [AFFINITY='&lt;affinity&gt;'] [JOB_AFFINITY='&lt;job affinity&gt;'] [ASYNC=true/false] [GENERATE_IIDS_FIRST=true/false] [ALLOW_MULTIPLY=true/false] [MAX_NODES=&lt;number&gt;] [MAX_WORKERS_PER_NODE=&lt;number&gt;]];</pre>	<pre>1) migratef P7.migrateFtest4().ig20 ; 2) migratef P1@DC1.migrateFtest4() from HIS_DB using ('select patient_id from invoice where balance=12894'); 3) migratef P7.migrateFtest4().('1', '2','3');</pre>



Command Name	Description	Syntax	Example
		<p>2) migratef          &lt;LUT&gt;[@&lt;DC&gt;].&lt;function&gt;() from          &lt;db_interface&gt; using          ('&lt;SQL&gt;') [WITH          [AFFINITY='&lt;affinity&gt;']          [ASYNC=true/false]          [GENERATE_IIDS_FIRST          =true/false]          [ALLOW_MULTIPLY=true/false]          [MAX_NODES=&lt;number&gt;]          [MAX_WORKERS_PER_NODE=&lt;number&gt;]]];</p> <p>3) migratef          &lt;LUT&gt;[@&lt;DC&gt;].&lt;function&gt;().(&lt;Instance 1,          Instance 2, etc...&gt;)          [WITH          [AFFINITY='&lt;affinity&gt;']          [ASYNC=true/false]          [GENERATE_IIDS_FIRST          =true/false]          [ALLOW_MULTIPLY=true/false]          [MAX_NODES=&lt;number&gt;]          [MAX_WORKERS_PER_NODE=&lt;number&gt;]]];</p>	

**Note:** A migration row summary appears at the end of migration with the following information:

- ADDED
- UPDATED
- UNCHANGED
- FAILED
- TOTAL
- DURATION\_SECONDS

For example, if a node went down while the migration was running, and therefore, not all instances were migrated.

**Note:** **Migrate All** is not supported for the following LUs:

- When the LU's root table has multiple populations.
- When the LU's root table has a root function. For such cases, you can use an instance group or query in order to execute the migration.

**Note:** MAX\_SOURCE\_QUERIES\_GROUPING parameter of config.ini defines the maximum value of OR statements in prepared statement query in case of many parents, related to one child. This value affects the WHERE clause length of the query of the tables population and its execution time. When the value is too large, the query might be rejected by the source DB. However, if the value is too small, it will slow the system by running too many queries.

## 4.6 Batch Processing Monitoring Commands

Command Name	Description	Syntax	Example
batch_list	Lists the history of all batch processes.  By default, lists all active /in progress batch processes.	batch_list [STATUS='<status>' [FROM_DATE='<from date>' [TO_DATE='<to date>']] [FILTER=<filter criteria>];	batch_list STATUS='ALL' - list the history of all the batch processes  batch_list status = 'ALL' FILTER = 'sync_instance';-  - list the history of all migrate processes. This command returns the same results as migrate_list STATUS = 'ALL'; command.  batch_list status = 'ALL' FILTER = 'sync_instance PATIENT'; - list the history of all migrate process into PATIENT LU.  STATUS - run with STATUS="" in order to get the list of supported statuses  FROM/TO_DATE - support DATE_FORMAT/DATETIME_FORMAT, according to the configuration in config.ini

Command Name	Description	Syntax	Example
			<p>FILTER- filtering the batch processes. The filter parameter needs to be populated by a String, included in the Fabric Command of the batch process.</p> <p>Note: the filter supports regex.</p>
batch_summary	Summary with all informative data on a given process id. The summary information displays the followings levels: node, DC, Cluster.	batch_summary '<batch id>';	<p>batch_summary '4bf0557d-182d-4489-9ec9-45be2601d006';</p> <p>Provides a summary on the batch id '4bf0557d-182d-4489-9ec9-45be2601d006'.</p>
batch_details	Show instances status for a given migrate id (start date, end date, running node). In case of failure, shows the details of the error message.	<p>batch_details '&lt;batch id&gt;'</p> <p>[STATUS='&lt;status&gt;']</p> <p>[ENTITIES='&lt;entity 1,entity 2,...&gt;']</p> <p>[AFFINITY='&lt;Affinity&gt;']</p> <p>[LIMIT='&lt;limit&gt;']</p> <p>[SORT_BY_PROCESS_TIME='&lt;true/false&gt;'];</p>	<p>batch_details '161f9717-bd93-4882-a3aa-7b58c1f61b27';</p> <p>Provides this list of instances of batch id '161f9717-bd93-4882-a3aa-7b58c1f61b27'</p> <p>STATUS - define STATUS=" in order to get the list of supported statuses.</p> <p>INSTANCES - List of instances separated by comma</p> <p>AFFINITY - DCs or nodes.</p> <p>LIMIT - default LIMIT is defined in config.ini if no LIMIT is provided as an argument.</p> <p>SORT_BY_PROCESS_TIME - if set to true, shows only the entities with the highest process time. <b>If set, ignore all other parameters.</b></p> <p>Note: The maximum number of instances, returned by this parameter, is defined in SLOWEST_PROCESSED_STATS_COUNT parameter of</p>

Command Name	Description	Syntax	Example
			config.ini file. The default value is 10 instances.
batch_units	<p>List of batch processes units at work.</p> <p>return: node id, batch process id, entity id, LU type, Duration (ms), Exeid, Command, Notes</p>	migrate_in_prcoess;	migrate_in_process;

## 4.7 Migrate Monitoring Commands

Fabric also supports the migrate commands for compatibility mode. Behind the scenes, Fabric activates the batch command when running migrate commands.

Command Name	Description	Syntax	Example
migrate_list	<p>Lists the history of all migrate commands.</p> <p>By default, lists all active /in progress migrates.</p>	<p>migrate_list</p> <p>[STATUS='&lt;status&gt;'</p> <p>[FROM_DATE='&lt;from date&gt;'</p> <p>[TO_DATE='&lt;to date&gt;']];</p>	<p>migrate_list STATUS='ALL' - list the history of all the migrate commands</p> <p>STATUS - run with STATUS="" in order to get the list of supported statuses</p> <p>FROM/TO_DATE - support DATE_FORMAT/DATETIME_FORMAT, according to the configuration in config.ini</p>
migrate_summary	<p>Summary with all informative data on a given batch id.</p> <p>The following information is returned for the batch id:</p> <ul style="list-style-type: none"> <li>Level-Node/DC/Cluster</li> <li>Name</li> <li>Status</li> </ul>	migrate_summary '<migrate id>';	<p>Migrate_summary '161f9717-bd93-4882-a3aa-7b58c1f61b27';</p> <p>Provides a summary on the migrate id 161f9717-bd93-4882-a3aa-7b58c1f61b27</p>

Command Name	Description	Syntax	Example
	<ul style="list-style-type: none"> <li>• Start time</li> <li>• End time</li> <li>• Duration</li> <li>• Remaining duration</li> <li>• Remaining</li> <li>• Total</li> <li>• Failed</li> <li>• Unchanged</li> <li>• Added</li> <li>• Updated</li> <li>• %Completed</li> <li>• Ent./sec (pace)-latest execution of entities per second</li> <li>• Ent./sec (avg.)-average number of executed entities per sec</li> </ul>		
migrate_details	Shows instances status for a given batch id (start date, end date, running node).In case of failure, shows the details of the error message.	migrate_details '<migrate id>' [STATUS='<status>' INSTANCES='<instance 1,instance2,...>' [AFFINITY='<Affinity>']] [LIMIT <limit>];	<p>Migrate_details '161f9717-bd93-4882-a3aa-7b58c1f61b27';</p> <p>STATUS - define STATUS=" in order to get the list of supported statuses.</p> <p>INSTANCES - List of instances separated by comma</p> <p>AFFINITY - DCs or nodes.</p> <p>SORT_BY_SYNC_TIME - if true, show only the entities with the highest sync time. If set, ignore all other parameters.</p> <p>Note: the maximum number of instances, returned by this parameter, is defined in SLOWEST_PROCESSED_STATS_COUNT parameter of</p>

Command Name	Description	Syntax	Example
			config.ini file. The default value is 10 instances.  LIMIT - default LIMIT is defined in config.ini if no LIMIT is provided as an argument.
migrate_in_process	List of instances at work on migrate commands.  return: node id, migrate id, instance id, LU type, time at work (ms), Exeid, Notes	migrate_in_prcoess;	migrate_in_process;

**Note:** All batch process Cassandra tables have a TTL (time to leave) of one week.

## 4.8 Resume (Retry) Batch Process

**Note:** If the batch process was not completed before the retry command, Fabric gets the list of instances from the source DB. If the batch process was completed before the retry command, Fabric gets the list of failed entities from batchprocess\_entities\_errors Cassandra table.

Command Name	Description	Syntax	Example
batch_retry	Resumes the previous batch process by processing all failed and not handled entities if the batch process was not completed, or retry only failed entities in case the previous batch process already completed.	batch_retry '<batch id>';	batch_retry '161f9717-bd93-4882-a3aa-7b58c1f61b27';

---

## 4.9 Resume Migrate

### 4.9.1 General

Fabric also supports the migrate commands for compatibility mode. Behind the scenes, Fabric activates the batch command when running migrate commands.

Command Name	Description	Syntax	Example
migrate_resume	Resumes a migrate command by processing all failed or not handled instances on previous executions.	migrate_resume '<migrate id>';	Migrate_resume '161f9717-bd93-4882-a3aa-7b58c1f61b27';

## 4.10 Cancel Batch Command

### 4.10.1 General

Migrate commands can be canceled using the cancel batch command.

In Fabric version 4.1 and higher, you can cancel migrate also by using the stopjob command.

Command Name	Description	Example
Cancel batch	Cancels the last started migrate for which the current node is its coordinator.  (Needs to be executed from the node that started the operation).	Cancel batch;
Cancel batch '<batch id>'	Cancels the batch with the defined batch id.  (no need to run from the coordinator node).	Cancel batch '568114fe-9ec8-4c9e-af11-6e3348eff6e9'

**Note:** The migrate status does not change when cancelling the migration process (run cancel batch command) in the middle of the process.

## 4.11 Security Definitions

### 4.11.1 General

Fabric database security definitions are saved under k2auth keyspace within 4 tables. Those definitions are being validated whenever a user is trying to access Fabric database through console, web service or any other interface. The permissions can be set in the LU type level or instance level.

**Note:** If you don't use a user\password when logging in to the Fabric database, the anonymous user will be used.

### 4.11.2 K2auth Tables

Table Name	Description
User Credentials	Holds Fabric database users and their roles. One user may have several roles.



Table Name	Description
Roles	Holds all type of roles.
Credentials	Holds token definitions for each role.
Permissions	Holds permission set for each role and method.

**Note:** You can set `DISABLE_LUI_AUTH` parameter of `config.ini` to 'true' to avoid an authentication of a user on the instance level (by default- this parameter is 'false').

### 4.11.3 Get Users Information from Cassandra

You can edit `SYNC_CASSANDRA_SYSTEM_AUTH` parameter of `config.ini` file to skip any interaction with Cassandra when adding/editing/removing users (except for actual authentication). By default, this parameter is set to 'true' and Fabric gets the users information from Cassandra. If you set this parameter to 'false', Fabric skips any interaction with Cassandra for users as explained above.

### 4.11.4 Add Users

Before creating any users, consider updating Cassandra's security settings. The following change prevents unauthorized users from manipulating security rules by directly updating the security tables under the `k2auth` schema:

Edit `/usr/local/k2view/apps/cassandra/conf/cassandra/cassandra.yaml`:

`authenticator: PasswordAuthenticator.`

`authorizer: CassandraAuthorizer`

Stop Cassandra

Start Cassandra

In order to create a new Fabric user:

1. Run `CREATE USER testUser WITH PASSWORD 'testUser' NOSUPERUSER|SUPERUSER`
2. Run `Assign role <Role_name> to user <User name>`

### 4.11.5 Add Roles & Token

To add the above mentioned role and assign it with a token (which will be used for executing K2View Fabric web services), you can use one of the following methods:

- Run the commands:
  - ◆ Under k2auth keyspace:
    - `INSERT INTO credentials (tok) VALUES ('k2view01')`
    - `INSERT INTO roles (name, description) VALUES ('SuperTester', 'Super Tester')`
    - `INSERT INTO credentials (tok, roles, user) VALUES ('k2view01', {'SuperTester'}, 'testUser')`
  - ◆ Under k2view key space:

Command	Description	Example(s)
Create token <token_name> [user 'user_comment'  ]	Creates a new token with or without a user comment.	Create token 'k2view01' user 'testUser'
Create role <role name> description ['<role description>']	Creates a role for Fabric users.	Create SuperTester role description 'new role for super user';
Assign role <Role name> to token <Token_name>	Assigns a token for a role.	assign role SuperTester to token k2view1

### 4.11.6 Drop

To drop a user, role or token, run the following command after logging in to Fabric:

Command	Description	Example(s)
drop user <user name>	Drops the user from Fabric.	drop user testUser
drop role <role name>	Drops a role from Fabric.	drop role SuperTester
drop token <token>	Drops a token from Fabric.	drop token k2view1

### 4.11.7 Grant Command

Grant commands are used to allow access to Fabric database for specified role\s.

These commands will insert records into the permissions table.

You can run several grant commands for the same LU type which will be appended.

#### 4.11.7.1 Grant All

Grant all is used to allow extensive Fabric database access for users, when using console or other available data viewers.

Command	Description	Example(s)
Grant all on * to <role name>	Allows the role to access all instances for all LU Types.	grant all on * to SuperTester
Grant all on <LUT> to <Role name>	Allows the role to access all instances for a specific LU Type.	grant all on CUST to SuperTester
Grant all on <LUT>.<Instance KEY> to <Role name>	Allows the role to access specific instances for a specific LU Type.	grant all on CUST.400000001 to SuperTester  grant all on CUST.400000002,CUST.400000003 to SuperTester

#### 4.11.7.2 Grant Web Service (Method)

The Grant Method is also used to allow Fabric database access for users when using Web Services.

Command	Description	Example(s)
Grant <Web Service Name> on * to <role name>	Allows the role to use the web service on all instances for all LU Types.	grant wsTest on * to SuperTester
Grant <Web Service Name> on <LUT> to <Role name>	Allows the role to use the web service on all instances for a specific LU Type.	grant wsTest on CUST to SuperTester
Grant <Web Service Name> on <LUT>.<Instance KEY> to <Role name>	Allows the role to use the web service specific instances for a specific LU Type.	grant wsTest on CUST.400000001 to SuperTester  grant wsTest on CUST.400000002,CUST.400000003 to SuperTester
Grant ALL_WS on <LUT> to <Role name>	Allows the role to use all web services on all instances for a specific LU/All Type.	Grant ALL_WS on * to SuperTester
Grant <Web Service Name> to <Role name>	Allows the role to invoke the WS. This is required in order to apply security on WS, which does not access Fabric data.	grant wsHelloWorld to SuperTester

### 4.11.7.3 Grant Operation

The Grant Operation command enables Fabric users to perform a defined list of activities on Fabric database, including:

- REVOKE\_ROLE
- ASSIGN\_ROLE
- EDIT\_ROLE
- READ
- DROP\_LUTYPE
- DEPLOY
- MIGRATE

Command	Description	Example(s)
Grant <operation> on <LUT> to <Role>	Allows a role to deploy LUs from Fabric Studio.	Grant deploy on MS to SuperTester Grant ASSIGN_ROLE on MS to SuperTester Grant MIGRATE on CUST to SuperTester

### 4.11.8 Revoke Command

Revoke removes granted permissions.

- Syntax: Identical to grant commands except using “revoke” instead of “grant” and “from” instead of “to”.
- Examples:
  - ◆ revoke all on CUST.400000001 from SuperTester
  - ◆ revoke wsTest on CUST.\* from SuperTester
  - ◆ revoke wsTest on CUST WHERE CUSTOMER\_TYPE<>'A' from SuperTester

### 4.11.9 Check Permissions

Checks permission for user on given method.

- Syntax: `CHECK_PERMISSION FOR <USER> ON <OPERATION>;`.
- Examples:
  - ◆ `CHECK_PERMISSION FOR SuperTester on MIGRATE`

## 4.12 Describe Command

The describe command is used to query Fabric's metadata structure.

Entity can be one of the following: CATALOG, SCHEMA, TABLE, INDEX

Pattern is a dot delimited hierarchy with % and \_ used as wild cards. If pattern is not specified all the information is returned.

Command	Description	Example(s)
DESCRIBE CATALOG CATALOG.SCHEMA	Get a list of schemas available in the specified catalog.  Fabric supports two catalogs: 'common' (reference) and 'mdb' (micro database).	DESCRIBE CATALOG; -- get LU and reference schemas.  DESCRIBE CATALOG mdb.%; -- get only LU schemas.
DESCRIBE SCHEMA SCHEMA.TABLE	Get a list of tables available in the specified schemas.	DESCRIBE SCHEMA;  DESCRIBE SCHEMA customer; -- get a list of tables in the customer LU schema  DESCRIBE SCHEMA customer.k%; -- get a list of tables starting with 'k' in the customer LU schema
DESCRIBE TABLE SCHEMA.TABLE	Get a list of columns for the specified table.	DESCRIBE TABLE; -- get the columns of all the tables in the system  DESCRIBE TABLE customer; -- get the columns of all the tables in the customer schema  DESCRIBE TABLE customer.address%; -- get the columns of all tables starting with address  DESCRIBE TABLE customer.address_billing; -- get the columns of the customer.address_billing table
DESCRIBE INDEX SCHEMA.TABLE	Get the list of indices for the specified table	DESCRIBE INDEX; -- get all system indices

Command	Description	Example(s)
		<p>DESCRIBE INDEX customer; -- get all indices in the customer schema</p> <p>DESCRIBE INDEX customer.address%; -- get the indices of all tables starting with address</p> <p>DESCRIBE INDEX customer.address_billing; -- get the indices of the customer.address_billing table</p>

## 4.13 List Command

Return a list of defined entities.

Command	Description	Example(s)
<p>List LU_TYPES/LUT [COUNT=&lt;'Y'/'N'&gt;] [LU_NAME=&lt;'NAME'&gt;]]</p> <p>[STORAGE=&lt;'Y'/'N'&gt; [LU_NAME=&lt;'NAME'&gt;]]</p>	<p>List the logical unit types.</p> <p>Options to get:</p> <ul style="list-style-type: none"> <li>■ Selective list of LU_NAMES</li> <li>■ Number of instances on each LU_TYPE</li> <li>■ COUNT - valid values are 'Y'/'N' (default is 'N'). If the COUNT is populated by 'Y', additional column will be added to the output. This column holds the number of all instances in fabric for each LU deployed. If LU_NAME is populated as well, the output will result in one row with the LU name and the number of its instances.</li> <li>■ STORAGE - valid values are 'Y'/'N' (default is 'N'). If this parameter is populated by 'Y', additional column will be added to the output. This column holds the storage for each LU deployed. If LU_NAME is populated as well, the output will result in one row with the LU name and its storage type. The values, displayed for the storage types are:</li> </ul>	<p>list LU_TYPES;</p> <p>list LUT count='Y' LU_NAME = 'PATIENT_DEMO_SOR' storage='Y';</p>

Command	Description	Example(s)
	<ul style="list-style-type: none"> <li>◆ Default - inherit the storage type from the config.ini file</li> <li>◆ Cassandra - store the instances in the Cassandra DB</li> <li>◆ None - the get command always gets the instance from the source DB. The instance is not stored in the Cassandra.</li> </ul>	
List DB_SOURCES	List the DB interfaces	list DB_SOURCES;
List WS	<p>List all Web Service methods. The following information is returned for each WS:</p> <p>Name</p> <p>Category</p> <p>Version</p> <p>Deployed - the latest deployment time for each WS.</p>	list WS;
List INSTANCE_GROUPS/IGS	List all instance groups	list INSTANCE_GROUPS;
List ENVIRONMENTS/ENVS	List all environments in the system	list ENVS;
List BROADWAY_FLOWS/BF	list all existing Broadway flows for all LUTypes with their inputs and outputs	List BF;

## 4.14 Delete Instances Commands

You can use the delete instance command to delete one or more instances from Fabric.

Command	Description	Example(s)
delete instance <LUT>.<instance_id>	Deletes one instance from Fabric.	delete instance CUST.102523826;
delete instance <LUT>.<instance_id>,<LUT>.<instance_id>	Deletes multiple instances from Fabric.	delete instance CUST.117035495, CUST.104680433, CUST.106673256;
Delete instance if not exist <LUT>	Deletes all instances that do not exist in the source system.	delete instance if not exist CUST

## 4.15 Queries Helpers

An SQL statement can be preceded by the keyword **EXPLAIN** or by the phrase **EXPLAIN QUERY PLAN**. This modification causes the SQL statement to behave like a query and to return information about how the SQL statement will operate if the **EXPLAIN** keyword or phrase is omitted.

The output from **EXPLAIN** and **EXPLAIN QUERY PLAN** is intended for interactive analysis and troubleshooting only.

Command	Description	Example(s)
EXPLAIN <SQL>	When the EXPLAIN keyword appears by itself, it causes the statement to behave as a query that returns the index file query it would have used to execute the command had the EXPLAIN keyword not been present.	EXPLAIN SELECT * FROM YP_CUSTOMER WHERE STATUS='O'
EXPLAIN QUERY PLAN <SQL>	The EXPLAIN QUERY PLAN SQL command is used to obtain a high-level description of the strategy or plan that is being used in order to implement a specific SQL query. Most significantly, EXPLAIN QUERY PLAN reports on the way in which the query uses database indices.	EXPLAIN QUERY PLAN SELECT * FROM YP_CUSTOMER WHERE STATUS='O'



## 4.16 Search/CDC

Command	Description	Example(s)
Search	<p>A search command provides the ability to run cross LUI search. Elastic Search command syntax should be used.</p> <p>Usage:</p> <p>SEARCH ltype=&lt;LUT_Name&gt; TABLES=&lt;tables names&gt; '&lt;Search Query&gt;';</p>	<pre>search ltype=OracleLu2 tables=PATIENT2{   "query": {     "query_string": {       "fields":         ["FIRST_NAME"],       "query":         "Moshe"     }   } }</pre>
cdc_republish_instance	<p>Republish instance cdc data</p> <p>Usage:</p> <p>cdc_republish_instance &lt;LUT_NAME&gt;.&lt;INSTANCE_ID&gt; [tables='&lt;table1,tables2...&gt;'] [types='&lt;type1,type2...&gt;'];</p>	<pre>cdc_republish_instance OracleLu.1 tables='tbl1,tbl2' types='type1,type2' ;</pre>

## 4.17 Broadway

Running a broadway flow

Command	Description	Example(s)
broadway	<p>Running a broadway flow.</p> <p>broadway &lt;LUT&gt;.&lt;FLOW_NAME&gt; [param1=value1, param2=value2...]</p>	<pre>Broadway PATIENT.Flow1 A=10, B=20</pre>

## 4.18 Other

Additional commands that can be used after logging in to Fabric are described in the following table.

Command	Description	Example(s)
version info	Supplies Fabric version installed on the server.	version info
ps	Shows the task IDs of all the instances that are currently in sync process on the specific node, and the exact population it working on and for how long.  Can run only by a user with admin role.	Ps;
Kill	Kill a task ID for an instance in sync on a specific node.  Can run only by a user with admin role	kill 'task id';
TEST_CONNECTION	TEST_CONNECTION provides connection status for a given interface according to the active environment	TEST_CONNECTION DbInterface='DB_HIS';
Release	RELEASE [<LU_NAME>]- use to detach lui from session	Release;  Release LU1;  Release LU1,LU2;

## 4.19 Reference Tables Commands

Command	Description	Example(s)
REF_SYNC [LU_NAME='lu name'] [TABLES='ALL' or '<table 1,table 2,etc...>'] [FORCE=true/false];	Start sync job for the specified common tables.	REF_SYNC TABLES='ALL';
REF_SYNC_WAIT [LU_NAME='lu name'] [TABLES='ALL' or '<table 1,table 2,etc...>'];	Wait for list of tables to be synced or transaction to be completed on the session, until a pre-defined timeout. Should run after REF_SYNC command or after insert/delete/update on the common table.	REF_SYNC_WAIT TABLES='ALL';
REF_STATUS [LU_NAME='lu name'] [TABLES='ALL' or '<table 1,table 2,etc...>'] [SCOPE='table' or 'population'];	REF_STATUS provides the tables sync status for the specified reference tables across all nodes. In addition, it provides the 'Current Session Transaction', i.e. the status of the	REF_STATUS;  REF_STATUS TABLES='ALL' SCOPE='population';

Command	Description	Example(s)
	<p>latest transaction, executed on the table in the current session.</p> <p>Default scope is table.</p> <p>Supported statuses:</p> <p>WAITING_FOR_SYNC</p> <p>Request for sync was issued but sync didn't start yet</p> <p>IN_SYNC</p> <p>In sync process</p> <p>READY</p> <p>Sync completed</p> <p>READY_WITH_BACKLOG</p> <p>Sync completed, be notified that there are still messages waiting in the backlog queue</p> <p>IN_BACKGROUND_SYNC</p> <p>Sync in background is currently running</p> <p>Current Session Transaction values:</p> <p>STARTED</p> <p>The transaction has started in the current session, but is not committed yet.</p> <p>COMMIT IN PROGRESS</p> <p>COMPLETED</p> <p>Commit is completed</p> <p>ROLLBACK IN PROGRESS</p> <p>ROLLED BACK</p> <p>Rollback is completed</p>	

## 4.20 Parser Commands

- The mechanism creates a job for each parser execution. If the parser is set to run automatically, a new record is created in k2\_jobs table when the parser is deployed to Fabric server. If the parser is set to run manually, a k2\_jobs record is created for the parser when it is executed.

- By default, Fabric allocates one available node to execute the parser.
- However, you can populate the 'Affinity' attribute for the parser execution. You can populate the affinity by either one of the following options:
  - ◆ IP address of a Fabric node
  - ◆ Data Center (DC) Name
  - ◆ Node identifier- a logical identifier a Fabric node or a group of several Fabric nodes
- You can also set several values, separated by comma, for the affinity.
- When you populate the affinity for a parser, Fabric allocates an execution server based on the populated affinity. If the DC name or the Logical node name contain several Fabric nodes, one of those nodes will execute the parser.
- When affinity is not populated, Fabric distributes the parsers between the different nodes of Fabric cluster.
- From Fabric version 4.1 and higher, the following additional interfaces are supported as input to the parsers:
  - ◆ HTTP
    - In order to set the HTTP interface, user should set the required host, port and protocol on new interface in Fabric Studio. The protocol can be either HTTP or HTTPS.
    - The HTTP connection can run only manually.
    - In contrast to SFTP connection, HTTP is different in the way it needs to be re-run on the required URL every time user wants to get the updated data stream.
  - ◆ JMS
    - Implementor can assign JMS consumer to subscribe above JMS queue/topic. By adding JMS interface and giving the relevant properties on Studio, new JMS consumer will start to subscribe. Two types of data are currently supported: TextMessage and BytesMessage.
    - Implementor can choose the preferred JMS provider on Studio. The following options are supported: ActiveMQ, RabbitMQ or MQ.
    - In case a new provider should be supported, implementor will have to assign the provider class name and provide path to the provider .bindings file.
    - Currently listening to JMS is supported only from Parser.
    - Inside the parser, implementor should use getMsg() function in order to get the JMS message. Then, he can get the message id and value and add the relevant logic that needs to be executed on each message.

- In contrast to SFTP interface, JMS interface is running in one thread that brings the messages and also executes the given logic. There is no additional parser to handle the implementor logic.
  - JMS interface was also added to the Environment functionality. It is possible to set a different parameter for each environment.
- ◆ KAFKA
- Implementor can assign Kafka consumer to subscribe Kafka topic. By adding Kafka interface and giving the Kafka bootstrap servers, topic name and group id on Studio, new Kafka consumer will start to subscribe. Currently two types of data are supported: String and ByteBuffer.
  - Inside the parser, implementor should use getMsg() function in order to get the Kafka message, then get the message key and value and add the relevant logic that needs to be executed on each message.
  - In contrast to SFTP interface, Kafka interface is running in one thread that brings the messages and also executes the given logic. There is no additional parser to handle the user logic.
  -

### 4.20.1 Add a New Configuration File- node.id

- A new configuration file was added to config directory of Fabric server- node.id.
- You can populate a logical identifier for the node on the node.id file.
- From Fabric version 5.2 and higher, it is possible to define cluster\_id on this file, in order to refer several Fabric clusters to the same Cassandra cluster. For example: cluster\_id:cluster1.
- Several nodes can have the same logical name in their node.id.
- The logical name of node is used for the population of the affinity attribute.
- The configuration of node.id file is optional.
- From Fabric version 4.1 and higher, it is possible to define priority for each logical/physical node name. In node.id file, you can set the number of threads that will be allocated for each node. For example, if node.id for a given node will be defined as:
  - ◆ A:2
  - ◆ B:2
  - ◆ C:6
- When there are 10 threads in the pool for this node, it means any job using logical node C as an affinity will get higher priority on this node, as 6 out of the 10 threads will be allocated to this job.

### 4.20.2 Parser Command

Command	Description	Example(s)
STARTPARSER <luName> [WITH AFFINITY='<affinity>']  (order of values of affinity is important)	Starts the parser on the local machine only for the LU type by name.	STARTPARSER CUST WITH AFFINITY='10.21.1.120'
STARTPARSER <luName><parserName>[WITH AFFINITY='<affinity>'] [ARGUMENTS='{<p1>:"v1",<p2>:"v2",...}'];	Starts the parser on a specific DC.  Affinity and Arguments are optional inputs.  Affinity: A comma separated list of DCs, IPs, and Node IDs combined.	STARTPARSER CUST pars1 WITH AFFINITY='ATL'
STARTPARSER <luName> <parserName>[@<dc>] [WITH UID='<uid>' [AFFINITY='<affinity>']] [ARGUMENTS='{<p1>:"v1",<p2>:"v2",...}'];  (order is important)	Runs the matching parser in the specified <lut> according to <parserName> and <uid>. Affinity and Arguments are optional inputs.  *Runs the specific parser <parserName> in the specified <lut>.	STARTPARSER CUST Pars1 WITH UID= '1' AFFINITY='DC1', '10.21.1.121'

Command	Description	Example(s)
	* UID: unique ID for parser that consists of a combination of LuName, parserName and affinity	
STOPPARSER <luName>	This command stops all the parsers under the LU.	STOPPARSER CUST
STOPPARSER <luName> <parserName>	This command stops the specific parser under the LU.	STOPPARSER CUST PARS1
STOPPARSER <luName> <parserName> WITH UID='<uid>'	This command stops the specific parser UID under the LU.	STOPPARSER CUST PARS1 WITH UID='<UID>'
SYNC_RECURRING_PARSER <lut>	Only for a recurring parser. Syncs all parser under the LU.	SYNC_RECURRING_PARSER CUST
SYNC_RECURRING_PARSER <luName> <parserName>	Only for a recurring parser. Syncs all the parsers with name equals to <parserName> under the LU.	SYNC_RECURRING_PARSER CUST PARS1
SYNC_RECURRING_PARSER <luName> <parserName> WITH UID='<uid>'	Only for a recurring parser. Syncs the specific parser <parserName> under the LU that matches the given <uid>	SYNC_RECURRING_PARSER CUST PARS1 WITH UID='<UID>'
RESTARTPARSER <luName>	Restart all the parsers under the LU.	RESTARTPARSER CUST
RESTARTPARSER <luName> <parserName>	Restart a specific parser under the LU.	RESTARTPARSER CUST PARS1
RESTARTPARSER <luName> <parserName> WITH UID='<uid>'	Restart a specific parser UID under the LU.	RESTARTPARSER CUST PARS1 WITH UID = '<UID>'
PARSERSTATUS [x days ago]	Will return the status for all parser jobs (not including the archived). That were created at most x days ago. When days not provided - Default is 1 day  Note: You can use the JOBSTATUS command and set the job type to PARSE to get the status parsers. See section 4.21.1 of this document.	PARSERSTATUS 2 days ago

Command	Description	Example(s)
PARSERSTATUS <luName> <parserName>	Will return the status for all parser jobs that match the given type and name including the archived.	PARSERSTATUS CUST PARS1
PARSERSTATUS <luName> <parserName> WITH UID='<uid>'	Will return the status of all matching rows including the archived.	PARSERSTATUS CUST PARS1 WITH UID='<UID>'

## 4.21 Job Mechanism Commands

The job mechanism is currently used for the following:

- Parsers execution
- Process files from predefined directories using SFTP or Local File System interface types
- Broadway
  - ◆ The ability to run a Broadway flow
    - Example: `startjob broadway_job name='<lu>.<flow>' [args='{"key":"value"}'];`
- User Jobs:
  - ◆ The ability to run light weight user functions without the need to execute parser.
  - ◆ Function should be defined as 'User Job' function type, it supports input parameters.
  - ◆ User jobs were added in Studio under LuType section in new 'Jobs' section.
  - ◆ Implementor has the option to run a function automatically on deploy or manually by 'startjob' command in console.
  - ◆ In order to start USER\_JOB function, provide lu\_name and function\_name in the job NAME:
 

```
startjob <USER_JOB> NAME='<lu_name.function_name>'
[UID='<uid>'] [AFFINITY='<affinity>'] [ARGS='<args>']
[EXEC_INTERVAL='<execInterval>'];
```
  - ◆ User job can be recurring job that runs automatically in every time interval or one time run by using 00:00:00 in the time interval.
- Process Jobs:
  - ◆ The ability to run a script or any other executable using the jobs mechanism.
 

```
startjob process NAME='/home/k2view/dateLog.sh'
UID='processJobtest' ARGS='{"0":"ARG 1 value","1":"ARG 2
value"}' EXEC_INTERVAL='00:00:03';
```



- The mechanism records all jobs activities in a new job table- k2\_jobs, which is created under k2systems keyspace of Cassandra.

#### 4.21.1 Job Commands

Command	Description	Example(s)
JOBSTATUS [x days ago]	When days are provided- returns the status for all jobs that were created during the last X days, including archived jobs. When days are not provided - returns all active (not archived) jobs.	JOBSTATUS;  JOBSTATUS 2 days ago;
JOBSTATUS <JOBTYP>	Returns the status of all running jobs according to jobType.	JOBSTATUS PARSE;
JOBSTATUS <JOBTYP> '<NAME>'	Returns the status of all running jobs that match the given type and name.	JOBSTATUS USER_JOB 'TDM.fnValidateAndRebuildRefTables';
JOBSTATUS <JOBTYP> '<NAME>' WITH UID='<UID>'	Returns the status of all jobs that match the given type, name and uid.	JOBSTATUS PARSE MyParser WITH UID='CUST-MyParser';
STARTJOB <JOBTYP> NAME='<name>' [UID='<uid>'] [AFFINITY='<AFFINITY=affinity>'] [ARGS='<args>'] [EXEC_INTERVAL='<execution interval>'];	<p>Starts the job using arguments.</p> <p>Affinity, UID and other Arguments are optional inputs.</p> <p>Jobtype - PARSE, INTERFACE, PROCESS, USER_JOB</p> <p>Name- job's name</p> <p>* args: String which represents a json to pass additional arguments to the start job command. e.g. {"parserName": "parserTest"} Execution interval as used in Fabric studio. The format is hh:mm:ss</p> <p>Affinity: Comma separated list of DCs, IPs and Node IDs</p> <p>EXEC_INTERVAL: definition of job scheduling execution interval, supports three formats:</p> <ul style="list-style-type: none"> <li>■ timestamp - 'yyyy-MM-dd HH:mm:ss' - a datetime to schedule job execution - one time  </li> <li>■ Time interval - 'HH:MM:SS' - run every X time</li> <li>■ Cron - crontab command combined. e.g. DC1,DC2...,IP1,IP2...</li> </ul>	STARTJOB PARSE NAME='CUST-MyParser' UID='CUST-MyParser' AFFINITY='10.21.1.85' ARGS={"parserName": "parserTest"} EXEC_INTERVAL='00:00:30';
STOPJOB <JOBTYP> NAME='<name>';	Stops all matching jobs with this name and type.	STOPJOB PARSE NAME=CUST-MyParser';
STOPJOB <JOBTYP> NAME='<name>' UID='<uid>';	Stops a specific job matching a UID.	STOPJOB PARSE NAME='CUST-MyParser' UID='CUST-MyParser';

Command	Description	Example(s)
RESTARTJOB <JOBTYPE> NAME='<name>';	Restarts all matching jobs with this name and type.	RESTARTJOB PARSE NAME='CUST-MyParser';
RESTARTJOB <JOBTYPE> NAME='<name>' UID='<uid>';	Restarts a specific job matching a UID.	RESTARTJOB PARSE NAME='CUST-MyParser' UID='CUST-MyParser';
RESUMEJOB <JOBTYPE> NAME='<name>' UID='<uid>';	Resumes a specific matching job. This command applies only to an existing job.	RESUMEJOB PARSE NAME='CUST-MyParser' UID='CUST-MyParser';
updatejob <jobType> NAME='<name>' [UID='<uid>'] [AFFINITY='<affinity>'] [ARGS='<args>'] [EXEC_INTERVAL='<execution interval>'] [RESET_END_TIME=true/false];	<p>Updates properties of existing job.</p> <p>AFFINITY: Comma-separated list of DCs and IPs combined. e.g. DC1,DC2...,IP1,IP2...</p> <p>* EXEC_INTERVAL: definition of job scheduling execution interval, supports three formats:</p> <p>timestamp - 'yyyy-MM-dd HH:mm:ss' - a datetime to schedule job execution - one time</p> <ul style="list-style-type: none"> <li>■ Time interval - 'HH:MM:SS' - run every X time</li> <li>■ Cron - crontab command</li> <li>■ * ARGS: String which represents a json to pass additional arguments to the update job command. e.g. {"parserName":"parserTest"}</li> </ul> <p>NOTE: in order to update cron job to one time job, use EXEC_INTERVAL="</p>	<p>UPDATEJOB PARSE NAME='CUST-MyParser' UID='CUST-MyParser' AFFINITY='10.21.1.85' ARGS={"parserName":"parserTest"} EXEC_INTERVAL='00:00:30';</p>

#### 4.21.2 Fabric V3.6- Cassandra Loader (Bulk Loader)

The Cassandra Loader will be used as a library to support massive writing operations to Cassandra DB. This library exposes a simple API to be invoked by multiple projects under Fabric solution.

The Cassandra loader is currently supported for iidFinder process and for parsers.

A new section was added to the config.ini and the iifConfig.ini files- cassandra\_cluster. This section was added for configuring the Cassandra driver connection.

The loader configuration for parsers is under the parsers section of the config.ini file. The setting of the config.ini is applied to all the parsers on Fabric node.

The iidFinder process always uses the Cassandra loader. A new section was added to the iifConfig.ini file for loader settings.

The following properties were added for the Cassandra loader:

Property Name	Description	Valid Values	Default Value
PARSER_WRITING_TYPE	Writing method into Cassandra- using Cassandra loader or JDBC driver for parsers	JDBC/LOADER	JDBC
EXECUTION_MODE	Loader execution mode	SINGLE/BATCH/ TOKEN_AWARE_BATCH/NOP	SINGLE
ASYNC	Statement execution mode- synchrony/asynchrony	true/false	true
ENABLE_FAILED_QUERIES_LOGGING	Enables query error logging when running the parser.	true/false	false
BATCH_SIZE	For modes BATCH/ TOKEN_AWARE_BATCH the number of statements in a batch.	Recommended 100	100
BATCH_THREADS_NUM	The number of threads for committing batch statements.	Recommended 5	5

- Cassandra loader execution modes:
  - ◆ SINGLE – every query submitted to the loader will be executed.
  - ◆ BATCH – in this mode the loader will execute statements in batches, the batch size is set by the BATCH\_SIZE property from config.ini or iifConfig.ini.
  - ◆ TOKEN\_AWARE\_BATCH - in this mode the loader will execute statements in batches, the statements will be grouped to batches by token to improve performance, the batch size is set by the BATCH\_SIZE property from config.ini or iifConfig.ini
  - ◆ NOP – in this mode, the loader doesn't execute the statements and discards them, should be used only for testing.

## 4.22 Cloning Fabric Session

Previously- when you needed to run activities on separate Fabric sessions- for example- get different instances- each one on a separate session- you had to define separate local fabric interfaces and open sessions using different interfaces.

Fabric 5.5 enables the user to clone the session into separate sessions using `openFabricSession` `UserCode` method:

```
openFabricSession(String interfaceName) : Connection
```

You need to send the name of the local Fabric interface as a parameter.

**Note:** if the `interfaceName` parameter is not defined as a local Fabric interface- an exception is thrown.

## 4.23 URL Rewriting

`Mod_rewrite` manipulates browser-submitted URLs and translates them to deliver content to the browser. This process takes place entirely at the server-side and without the browser's knowledge. The resulting content appears to have originated from the submitted URL.

For example:

A user may ask for `http://www.somesite.com/widgets/blue/`, but will really be given `http://www.somesite.com/widgets.php?colour=blue` by the server. Of course, the user will be none the wiser to this little bit of chicanery.

- How to configure:  
Copy `rewrite.config` file (sample file attached) to  
`$FABRIC_HOME\webserver\WEB-INF`
- The user has to have at least basic knowledge of rewrite rules/conditions and their different parameters in order to be able to use this functionality.
- Useful tutorials:  
<https://tomcat.apache.org/tomcat-8.0-doc/rewrite.html>  
[https://www.sitepoint.com/apache-mod\\_rewrite-examples-2/](https://www.sitepoint.com/apache-mod_rewrite-examples-2/)  
[http://helpful.knobs-dials.com/index.php/Apache\\_config\\_and\\_.htaccess\\_-\\_URL\\_rewriting](http://helpful.knobs-dials.com/index.php/Apache_config_and_.htaccess_-_URL_rewriting)

---

## 4.24 Salesforce Support

You can use Fabric as external source for Salesforce using oData protocol.

For configuration details, please use the [URL](#).

## 5 Fabric as System of Record

- This feature enables running single transaction - insert/insert or replace/update/delete - on a specific table of the instance ID, or on the common table, instead of synchronizing the entire instance id or common table from source.
- This way, the fabric can get transactions feeds and update the related instance IDs/commit tables accordingly.
- Start transaction - you must begin transaction before running the insert/update/delete commands.
- End transaction:
  - ◆ end/commit commands - commit the changes and end the transaction.
  - ◆ Rollback/exit/CTRL C/kill - rollback the changes and end the transaction.

### 5.1 Update LU Instance - Synchronic Mode

- The transaction must run on LUI level, i.e. you must get the LUI before running the insert/update/delete transaction.
- The get entity can be set inside or outside the transaction. However, if your transaction updates the LUI in the scope, you cannot detach schema and get a different instance until you commit or rollback your changes.
- It is possible to define tables without any population in the LU. These tables must be added to the LU schema, although these tables will not be linked to any parent table. They will not be populated by the sync of the entity, but can be populated by an external transaction like user job, parser, web service, or by using the console.
- Add new configurable parameter in config.ini per node to support lightweight transaction - OPTIMISTIC\_LOCKING. Supported Values:
  - ◆ NONE - this is the default value. The latest transaction overrides the instance ID.
  - ◆ QUORUM - the first transaction locks the instance ID. Latest transaction will fail until the transaction is committed (the commit requires a quorum).
  - ◆ LOCAL QUORUM - the first transaction locks the instance ID. Latest transaction will fail until the transaction is committed (the commit requires a local quorum on the DC).
- OPTIMISTIC\_LOCKING modes - Example:

- ◆ Fabric has 2 Data Centers:



- ◆ Transaction 1- run on Node1 (DC1)
  - ◆ Transaction 2- run on Node4 (DC2)
  - ◆ OPTIMISTIC\_LOCKING='NONE' => transaction2 (the latest transaction) overrides transaction 1
  - ◆ OPTIMISTIC\_LOCKING='QUORUM'=> transaction 1 locks the instance till the transaction is committed and updates at least 2 nodes of each DC
  - ◆ OPTIMISTIC\_LOCKING='LOCAL QUORUM' => transaction 1 locks the instance till the transaction is committed and updates at least 2 nodes of DC1
- You can begin transaction and update the LUI inside by the table populations, parsers, web services, user jobs, or using the fabric console.

#### Notes:

- You can write into a table with or without population. However, it is up to the implementation to verify that the data, populated by the population object when the instance ID is synced, does not conflict or override the changes of the table.
- You can view your changes from your session even before committing your changes.
- You can include several insert/update/delete commands on the instance ID in the same transaction.
- You can get and update only one instance ID per LU per transaction. For example- you can get PATIENT 1 and VISIT 100 in one transaction, but you are not allowed to get PATIENT 1 and PATIENT 2 in one transaction.
- Like previous Fabric versions, there is a write lock per LUI. It means that you can begin several transactions on one LUI only when you open the transactions in different nodes. You cannot begin several transactions on the same LUI on one node. When you try to get instance or begin transaction for instance which already has an open transaction on the same node, Fabric will wait for 10000 MS (configurable parameter) for the release of the transaction and then will throw error.

## 5.1.1 Synchronic Update of LUI- Examples

### 5.1.1.1 Implementation code - run transaction using fabric user code methods:

```
Db ci= db("fabric");
// Get the entity
ci.execute("get PATIENT_DEMO_SOR." + IID + "");
// Begin Transaction
ci.beginTransaction();
String SQL = "insert into INVOICE (PATIENT_ID, INVOICE_ID, ISSUED_DATE,
DUE_DATE, STATUS, BALANCE) values (?, ?, datet('now'), date('now', + '10
days'), 'OPEN', ?)";
Object[] params = new Object[] {IID, invID, balance};
ci.execute(SQL, params);
ci.execute("update invoice set status = 'O' where status = 'OPEN' ");
// Commit the transaction
ci.execute("commit");
```

### 5.1.1.2 Run Transaction Using the Fabric Console

```
// get the entity
get PATIENT_DEMO_SOR.22;
// Begin Transaction
begin;
// Run update command
update visit set physician = 'Dr. House' where physician is null;
// Commit;
commit;
```



## 5.2 Update LU Instance-Asynchronous Mode

- Support asynchronous mode of update/insert/delete transaction. This transaction is populated in the **delta table of the iidFinder** mechanism (the delta table is created under the k2staging keyspace). The iidFinder gets the transaction from the delta table and updates the LU instance.
- A new command was added to the fabric:
  - ◆ set async\_trx;
  - ◆ Valid values are true/false.
  - ◆ You must set the async\_trx to true before the beginning of the transaction to run asynchronous mode when updating LU instance.

**Note:** The transaction is not validated, but added to the delta “as is”.

## 5.3 Update Common Tables

- You can run transactions on common (reference) tables.
- Enhance the REF\_SYNC\_WAIT and REF\_STATUS commands to support transactions on common tables. See more details in section 4.19 of this document.
- The insert/insert or replace/update/delete commands must be wrapped by the BEGIN and COMMIT/ROLLBACK commands.
- Like LU table, you can define a common table without population. This table can be populated by a transaction.
- The transaction of the common table is done in asynchronous mode, i.e. you cannot view the updated data until you run the commit, and the fabric updates the common table.
- The transaction is sent to Kafka and is saved into the Kafka or the Cassandra, depending on its size:
  - ◆ A new configurable parameter in config.ini - TRANSACTION\_BULK\_SIZE- defines the maximum number of commands, sent by each bulk.
- For example:
  - ◆ TRANSACTION\_BULK\_SIZE= 1000
    - Run 2500 insert commands. Each bulk of 1000 commands it sent to Kafka, the commands are kept in the Cassandra, and the Kafka gets

the transaction ID. The 2500 inserts are divided into 3 transactions (1000 + 1000 + 500).

- Then, open a new begin and run 900 inserts. The 900 inserts are sent and stored in Kafka.
- Parallel transactions on common tables:
  - ◆ The first commit updates the table. The commit is initiated:
    - Short transaction- the user runs the commit command.
    - Large transaction- the commit is initiated internally for each bulk size, populated in the Cassandra.
- Common Snapshot:
  - ◆ When you run delete and then insert on a common table in the same trx, or you run delete command in the trx, the fabric creates a snapshot on the common table.

**Note:** You can still create a snapshot using population, and ref\_sync when the population includes a truncate of the table. The sync is a separate trx.

---

## 6 Fabric API Documentation

Use the following URL to view the API documentation of the deployed fabric version on the fabric server:

- <http://<IP address>:3213/static/doc/user-api/index.html>

---

## 7 Log Tracking

Any activity which is performed within Fabric database is either being printed into the log file, printed to the screen, or written into the log table. When executing a get command for a single instance, you can open the log file in order to view errors, warnings or messages, but when executing MPP operation like multiply instances migration, it is easier to view the errors from a log table. In addition, you can use the log table in order to generate reports.

---

### 7.1 Configuration File

Some parameters can be set within the log configuration file:

- File location:

config/logback.xml

For additional information, you may refer to

<http://logback.qos.ch/manual/configuration.html>

- Example for parameters:

- ◆ `<root level="INFO">`

- ◆ `<appender-ref ref="STDOUT" />`

`</root>`

This parameter is used in order to define whether the errors will be written to the screen, log file or/and log table.

In order to write the error to a log table, add 'DB' at the end.

`<logger name="com.k2view" level="INFO"/>`

This parameter defines the message level which should be written into the log. There are 4 levels (from least severe to most severe): DEBUG, INFO, WARN and ERROR.

With the above setting, DEBUG messages will not be written into the log as its level is lower than INFO.

**Note:** If you would like to print all K2View Fabric messages into the log, set this parameter to DEBUG which is the lowest level.

## 7.2 Log File

The log file is located at '/usr/local/k2view/logs/k2fabric.log'. This is also a configurable parameter in the above mentioned config file.

### 7.2.1 Add Connection Leaks to the Log File

SUSPECTED\_CONNECTION\_LEAK\_SEC parameter of config.ini file enables getting the information about connections which are suspected as leaked, into the error file. The default of this parameter is 0 (turned off). When this parameter is set to value greater than 0, a connection that is idle for this many seconds, will be suspected as leaked. The system will log this into k2fabric.log and show a stack trace of the getConnection() method in order to indicate the piece of code that is causing the connection leaked.

## 7.3 Log Table

The log table is located within the k2systemkeyspace. By default, messages are not being written into this table for performance reasons. If required, you can change the configuration file as mentioned above.

### 7.3.1 Table Structure

Node_id	The Cassandra node on which this error occurred (not relevant for single node environments)
Timestamp	
Error_id	Fabric database error code
Details	Error message parameters
Error_msg	Error message
Instance_id	
Lu_type	
Severity	Debug Info Warn Error
Table_name	If relevant, the table where the failure occurred

### 7.3.2 Error Code

The following Fabric database error codes are currently supported:

Error Code	Description
FABRIC0001	User Message
FABRIC 0011	General SQLite
FABRIC 0101	General Cassandra error
FABRIC 1000	General K2View Fabric info
FABRIC 1001	General K2View Fabric error
FABRIC 1011	Translation failed
FABRIC 1012	K2View Fabric Synchronization
FABRIC 1013	Record rejected
FABRIC 1014	Instance rejected
FABRIC 1015	Method not found
FABRIC 1016	Instance was not found in source system
FABRIC 5001	General Exception

---

## 8 Servlet Filters

A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource, or both. A servlet filter is configured to add functionality to the web service calls such as authentication, auditing, remote control and more.

The filter should only interact with implementation web services and not internal product calls which it will break.

To setup Tomcat servlet filters use the `WEBSERVER_FILTERS` parameter in the `config.ini` of Fabric server.

Below is an example for `WEBSERVER_FILTERS` population:

- `WEBSERVER_FILTERS=[`  
    `{"class":"com.custom.MyFilter","patterns":["/ws","/api/*"],"params":{"p1":"v1","`  
    `p2":"v2"} }, {"class":"com.custom.AnotherFilter"} ]`

**Notes:**

You need to add the jar file with the Filter class to Fabric Class Path.  
Patterns parameter is optional and will default to the app web services.  
Params parameter is optional.

---

## 9 Enable GZIP Compression of Web Service Response

Enable GZIP compression on web services response if it exceeds a predefined size.

The following parameters were added to config.ini file by Fabric 5.5-

- WEB\_SERVICE\_COMPRESSION. The default value is 'on'. Valid values: on, off, force.
- WEB\_SERVICE\_COMPRESSION\_MIN\_SIZE- the minimum amount of data of the web service response before the output is compressed. This parameter is checked when the WEB\_SERVICE\_COMPRESSION is set to 'on', the default value is set to 2048 KB.



## 10 Trace Mechanism to Fabric

Trace mechanism enables the tracing of Fabric internal operations by request. The trace mechanism creates tracing files.

The TRACE command controls the active traces in the system and supports the following options:

To get a list of available operations:

- trace operations;
- trace ops;
- List of available operations:
  - ◆ operation | category | base\_level
  - ◆ -----+-----+-----
  - ◆ connection | database | info
  - ◆ preparedStatement | database | info
  - ◆ sqlCommand | database | info
  - ◆ saveLuToCassandra | database | info
  - ◆ referenceTablesRefresh | database | info
  - ◆ fabricPrepareCommand | commands | info
  - ◆ fabricCommand | commands | info
  - ◆ deploy | commands | info
  - ◆ jobsHeartbeat | jobs | debug
  - ◆ jobExecutor | jobs | info
  - ◆ folderPoller | jobs | debug
  - ◆ sftpReader | jobs | debug
  - ◆ tryClaim | jobs | debug
  - ◆ jobsPoolExecutorGet | jobs | debug
  - ◆ jobsPoolExecutorSubmit | jobs | debug
  - ◆ webservice | solutions | info
  - ◆ sync | solutions | info
  - ◆ parser | solutions | info
  - ◆ custom | solutions | info
- The trace name is an alphanumeric value (cannot use a reserved sql keyword). The trace parameters are optional and include, among others, the following information:
  - ◆ Operations to monitor, resource to filter (for example billing DB), trace level.

You can do: get trace status, stop trace, and delete trace files as follow:

- Trace command syntax:  
`trace <TRACE_NAME> '[TRACE_PARAM=[TRACE_VALUES]]';...`
- TRACE\_PARAM is optional and can be populated by one of the following:
  - ◆ `operations=OP1,OP2,...` Which operations to monitor, default is none
  - ◆ `resources=RES1,RES2,...` Which resources to filter, default is no filter, all resources under an operation.
  - ◆ `level=LEVEL` Which level to use:  
`debug|verbose|info|warning|error` default is info.
  - ◆ `categories=CAT1,CAT2,...` Shortcut to turn on all operations under a category.
  - ◆ `syncWrite`- Perform synchronous writes, default is asynchrony write to files.
  - ◆ `maxCols=N` - How many columns to write out of the trace data, default is 10.
  - ◆ `maxColLen=N` - How many bytes to write out of each column, default is 256.
- Example:  
`trace <TRACE_NAME>`  
`level=debug;categories=databases;resources=billing,crm;operations=solutions;`  
`syncWrite;maxCols=3;maxColLen=128;`
- To get trace status
  - ◆ `trace status; -- All traces`
  - ◆ `trace status <TRACE_NAME>; -- Specific Trace`
- To stop a trace
  - ◆ `trace off; -- Stop all traces`
  - ◆ `trace off <TRACE_NAME>; -- turn off a specific trace`
- To delete a trace - Stop the trace if on and delete the trace files.
  - ◆ `trace remove <TRACE_NAME>;`
- Trace report located at FABRIC\_HOME/trace or can be view using Trace UI:  
<http://<Fabric Server>:3213/static/trace/trace.html>.
- To receive syntax for all commands related to trace options, type in CLQSH :  
`help trace`.

## 11 JMX - Java Management Extensions

New feature of monitoring Fabric using JMX.

The JMX will be enabled by default from version Fabric 3.3.

The user will be able to monitor Fabric metrics using jconsole locally or remotely.

In addition, there is a web service to access the metrics <http://localhost:3213/static/status/status.html>.

Some of the bold metrics that are monitored:

- Active and idle connections in the system for each DB interface
- Metrics related to common tables
- Web service open connections and call duration per web service
- Fabric commands success and failure count for each commandWorker threads in the system per worker type

### 11.1 Custom JMX Metrics

CustomStats adds the ability of defining JMX metrics on the project level.

The CustomStats can be invoked using the following UserCode methods:

- `public static void statsCount(String entry, String key, long value):`
  - › This method counts statistics such as events, bytes, counts etc. and exposes them as JMX counters.
  - › Parameters:
    - `entry` - the primary key for these statistics
    - `key` - the sub key for these statistics
    - `value` - the measured value for these statistics. The stats system will use this value to compute total (the accumulated total of this event value), last (the last value for this event), and the average (the average of this event value since the process was launched) parameters. In addition, the system measures the count (times the stat has been called) and the timestamp of the last call.
  - › Example-
    - `statsCount("met1", "met2", 2)`- this will create main key "met1", sub key "met2" and populate their value by 2.
- `public static AutoCloseable statsDuration(String entry, String key):`
  - › This method measures duration- the time between the method invocation and the invocation of the close method on the return object.

- › Parameters:
  - entry - the primary key for this statistic
  - key - the sub key for these statistics
- › Returns:
  - Call .close() on this object to indicate the end of the duration measurement
- › Example-
  - AutoCloseable sw = statsDuration("test1", "test2");
  - sleep(10);
  - sw.close();

**Notes:** You can view the JMX metrics using the following URL:  
<http://<IP address>:3213/static/status/status.html>  
A formal guide will be issued.

## 12 Fabric Auditing

- The auditing saves all activities, performed on Fabric, in k2\_auditing table under k2audit Cassandra keyspace.
- K2auditing contains the following information on each activity:
  - ◆ action – the type of activity, performed on Fabric. For example- CQLCommand, LOGIN, Web service name.
  - ◆ date – activity date without time
  - ◆ user
  - ◆ written\_at – activity date with timestamp
  - ◆ address – the address of the node where the activity was performed
  - ◆ params
  - ◆ protocol
  - ◆ query – for example- CQL query for CQLCommand
  - ◆ result- number of affected rows
  - ◆ session\_id
- The following parameters were added to config.ini to support auditing:
  - ◆ AUDIT
    - Valid values are 'OFF' and 'ON'. The default value is 'OFF'. To enable auditing on Fabric, set the AUDIT parameter to 'ON'.
  - ◆ AUDIT\_FILTER\_STRATEGY- you need to populate a full path to class defining the filter strategy, to allow/avoid audit of certain operations. This parameter is optional.
  - ◆ AUDIT\_PERSISTENCE\_STRATEGY- you need to populate a full path to class defining the persistence strategy to write the audit messages into Kafka, file, or other DB instead of Cassandra (default option). This parameter is optional.
    - The default value of this parameter is:  
AUDIT\_PERSISTENCE\_STRATEGY=com.k2view.fabric.session.auditing.persistence.CassandraBeanPersistence
    - Write the audit messages to Kafka- you need to populate this parameter as follows:  
com.k2view.fabric.session.auditing.persistence.KafkaBeanPersistence
    - Custom implementation-you need to populate the full path of your customization class. For example:
- AUDIT\_PERSISTENCY\_STRATEGY =  
com.k2view.external.fabric.audit.persistencies.SamplePersist

- You must restart the Fabric node after editing the audit parameters of the config.ini file.

**Note:** You can set different audit options for each Fabric node.

## 12.1 Custom Auditing

### 12.1.1 Build Custom Audit

- Create filter or persistence class (see detailed explanation below).
- Build artifacts for your class:
  - ◆ Open your code using IntelliJ IDE.
  - ◆ From IntelliJ: Build => Build Artifacts => Select desired module ( Or select All) => Build
- Save the artifacts into the following path:  
AuditCustomStrategies/out/artifacts/<module>/<module>.jar
- Place the artifacts in Fabric's ExternalJars directory.  
For example: \$K2\_HOME/ExternalJars
- Update \$K2\_HOME/config/config.ini – update  
AUDIT\_PERSISTENCY\_STRATEGY and AUDIT\_FILTER\_STRATEGY
- Restart your Fabric node.

### 12.1.2 Filter Activities for Audit

- Create a new class under the package  
"com.k2view.external.fabric.audit.filters". Implement the following Interface to define audit filter:
- com.k2view.fabric.session.auditing.filters.AuditingFilter;
- Below is an example of a filter class -  
com.k2view.external.fabric.audit.filters.SampleFilter. This class accepts auditing only when a user is "admin".

```
package com.k2view.external.fabric.audit.filters;

import com.k2view.fabric.common.Log;
import com.k2view.fabric.session.auditing.AuditBean;
import com.k2view.fabric.session.auditing.filters.AuditingFilter;

public class SampleFilter implements AuditingFilter {
    @Override
    public boolean filter(AuditBean auditBean) {
        /*
         * Here You should add your code
         * when return false - the bean will not be audited
        */
    }
}
```

```

        * when return true - the bean will be audited
        */
        /*
        if(auditBean.getUsername().equalsIgnoreCase("admin")) {
            Log.a(SampleFilter.class).info("Bean {} is passes the filter
layer", auditBean.toString());
            return true;
        }
        */
        /* if(auditBean.getAction().equalsIgnoreCase("GetCommand")){
            Log.a(SampleFilter.class).info("Bean {} is passes the filter
layer", auditBean.toString());
            return true;
        }*/

        /*else if(auditBean.getAddress().equalsIgnoreCase("10.21.1.105")){
            Log.a(SampleFilter.class).info("Bean {} is passes the filter
layer", auditBean.toString());
            return true;
        }*/

        if(auditBean.getUsername().equalsIgnoreCase("michal") ||
auditBean.getUsername().equalsIgnoreCase("mic")){
            Log.a(SampleFilter.class).info("Bean {} is passes the filter
layer", auditBean.toString());
            return true;
        }

        else {
            Log.a(SampleFilter.class).info("Bean {} isn't passes the filter
layer", auditBean.toString());
            return false;
        }

    }
}

```

- Update config.ini - populate the full path of your filtering class in the AUDIT\_FILTER\_STRATEG parameter. For example:
- AUDIT\_FILTER\_STRATEGY=com.k2view.external.fabric.audit.filters.Sample Filter.

### 12.1.3 Custom Persistency Strategy

- Create a new class under the package "com.k2view.external.fabric.audit.persistencies". Implement the following Interface:  
com.k2view.fabric.session.auditing.persistence.AuditBeanPersistence;
- Below is an example of a persistency class:  
com.k2view.external.fabric.audit.persistencies.SamplePersist. This class writes the audit operations into a file.

```
package com.k2view.external.fabric.audit.persistencies;

import com.k2view.fabric.common.Log;
import com.k2view.fabric.session.auditing.AuditBean;
import com.k2view.fabric.session.auditing.persistence.AuditBeanPersistence;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.Date;

public class SamplePersist implements AuditBeanPersistence {
    @Override
    public void persist(AuditBean auditBean) throws Exception {
        /*
         * Here you should add your way of how to persist the bean
         */
        Date d = new Date();

        String fileName="/home/k2view/AuditFiles/AuditFile.txt";

        BufferedWriter writer = new BufferedWriter(new FileWriter(fileName,
true));
        writer.write("Bean {} will not be persisted "+ auditBean.toString());
        writer.flush();
        writer.close();
        Log.a(SamplePersist.class).info("Bean {} will not be persisted",
auditBean.toString());
    }
}
```

- Update config.ini - populate the full path of your class into AUDIT\_PERSISTENCY\_STRATEGY parameter. For example:
- AUDIT\_PERSISTENCY\_STRATEGY =  
com.k2view.external.fabric.audit.persistencies.SamplePersist
- lid\_



---

## 13 Cleanup Procedures

You can clean up Fabric database: all LUs, a specific LU, or a single instance using the steps below.

---

### 13.1 Drop All LUs (keyspace)

In order to clean all data from Fabric database, you should execute the reset script located under: `/usr/local/k2view/fabric/scripts`. Use one of the following modes:

- Drop – drop the stored cassandra db  
`./reset.sh drop`
- Drop\_local – drop the local node storage (such as: `/dev/shm/`)  
`./reset.sh drop_local`
- Truncate - truncate all Fabric keyspaces  
`./reset.sh truncate`

---

### 13.2 Drop Specific LU (keyspace)

In order to drop a specific LU (keyspace), execute the following steps through command line:

1. Run `fabric`
2. Run `drop LUTYPE <LU TYPE>`  
Examples: `drop LUTYPE CUST`

---

### 13.3 Switching Between the Global Environments

To receive syntax for all commands related to Global Environment, type:

`help set_global environment`

The list of available commands is displayed.

Change the Global Environment by using the following command:

`set_global environment= '<environment name>'`

```
> help set_global environment;
|Topic
+-----+
|SET_GLOBAL ENVIRONMENT - Set the environment for the whole cluster|
|
|set_global environment '<environment name>'
|
(3 rows)
> set_global environment='ENV1';
(1 row affected)
```

## 13.4 Switching Between the Session Environments

**Note:** The Session Environment does not change the Global Environment. After exiting Fabric session and opening a new session, the opened Session Environment will be the Global Environment.

To receive syntax for all commands related to Session Environment, type:

`help set environment`

The list of available commands is displayed.

```
> help set environment;
|Topic
+-----+
|SET ENVIRONMENT - Set the environment for the current session|
|
|set environment='<environment name>'
|
|To reset the session environment:
|set environment=''
|
(6 rows)
```

Changed the session Environment by using the following command:

`set environment= '<environment name>'`

---

## 13.5 Checking Active Environment

You can check the active Environment, by using the following commands:

- Connect to Fabric.
- Run the command `set`. Active Environment name is displayed in the Status column.

---

## 13.6 Deploy Enviroments (From File)

- Description:  
Besides the option to deploy environments from the studio, there is an option to deploy the environments using the file generated by the studio.
- Syntax:  
DEPLOY ENVIRONMENTS FROM FILE '<file name>' – file name should include the full path in server
- Generating the deploy artifact:
  - ◆ From the Fabric studio, right click Environment and select Open Folder (file will be available only after Deploy from studio)
  - ◆ The file will be copied to the server before executing the offline deploy.

## 13.7 Setting Globals

- **Description:**  
From Fabric version 4.1 and higher, it is possible to override globals without deploying the implementation. The command can run only on globals defined as not final.
- **Syntax:**
- `set_global global '<LUT_NAME>.<PARAM_NAME>[=<PARAM_VALUE>]'`
- Setting PARAM\_NAME in LUT NAME with the value provided by PARAM\_VALUE.
- When PARAM\_NAME is given without '=<PARAM\_VALUE>' it will be set to the original value.
- When PARAM\_NAME is given with '<LUT NAME>.<PARAM\_NAME>=' it will be set to null.
- `set_global global '.*.<PARAM_NAME>[=<PARAM_VALUE>]'`
- Setting PARAM\_NAME on all LUs with the value provided by PARAM\_VALUE.
- When PARAM\_NAME is given without '=<PARAM\_VALUE>' it will be set to the original value.
- When PARAM\_NAME is given with '<LUT NAME>.<PARAM\_NAME>=' it will be set to null.
- **Remark:**  
Global that is defined on shared level as final, but also on LU level under the same name as not final, can be overridden.

## 13.8 Permissions

The following actions require special permissions:

Action	Permission
To deploy Environment	DEPLOY_ENVIRONMENTS
To set Environment for current session	SET_ENVIRONMENT
To set Global Environment	SET_GLOBAL_ENVIRONMENT

## 14 Web Admin

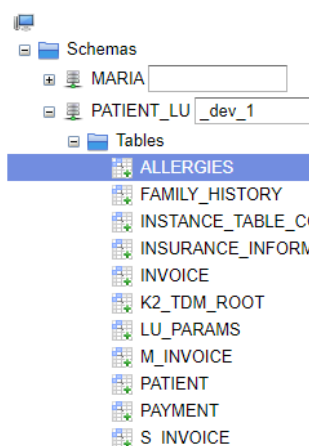
Web Admin is an interface for:

- Managing roles and security.
- Displaying the data that exists in Fabric database for auditing purposes.
- Performing data migration into Fabric.

To access the Admin page, open the **http://<fabric server IP>:3210/admin server IP>:3210/admin** URL and log in with your Fabric credentials.

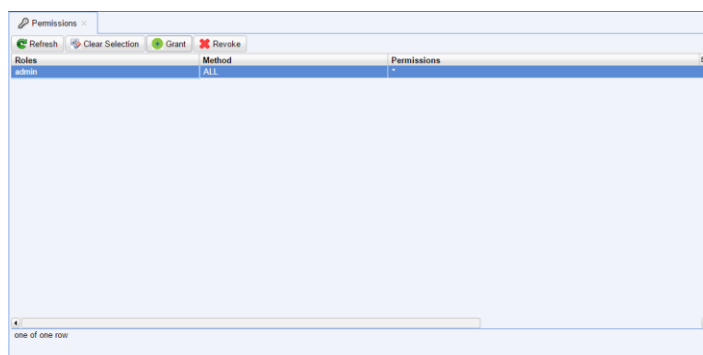
**Note:** When opening new admin tab, show login screen will not appear.

The tree displayed on the left side of the window shows the data in Fabric database, based on user permissions. The Security branch and its sub branches enables you to configure various security options.




### 14.1 Managing Permissions

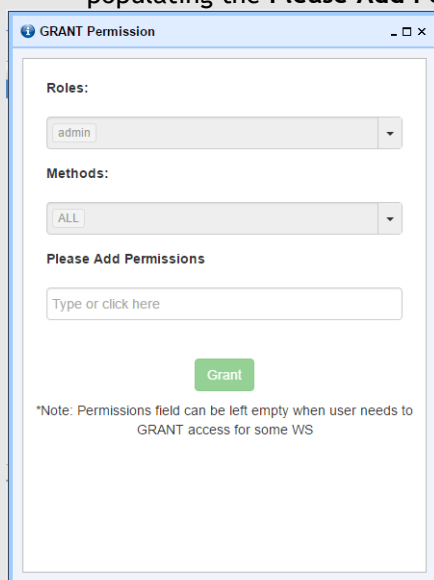
Click the  icon to display the **Permissions** tab. This tab enables you to grant and revoke permissions to a role.



The **Permissions** tab displays the following buttons:

- **Refresh:** Refreshes the permission list.
- **Clear Selection:** Clears the role selection in the *Grant Permission* window (see below).
- **Grant:** Click the  button to display the following window. This window enables you to assign the requested permissions. If a role is selected in the Permissions tab, the Roles and Methods fields are automatically set and disabled.

For methods that are not web services, the **Grant** option is available only when populating the **Please Add Permissions** field.



#### To assign permission to a role

- ◆ Select a role name in the Roles dropdown menu.
- ◆ Select the method(s) in the Methods dropdown menu. The selected method(s) determine(s) the functionality that the user can access in the user interface. For example, select READ for the user to be able to read data from Fabric, but not migrate any instances or deploy the LU into Fabric.
- ◆ In the Please Add Permissions field, specify the logical units or specific instances for which this role is permitted. Separate multiple permissions by a period (.). When entering multiple permissions, enter a permission followed by a period, press <Enter> and then enter the next one. When entering a specific logical unit instance, only that instance is allowed. For


example, you can enter CUST.4333 for a specific instance or CUST for all instances.

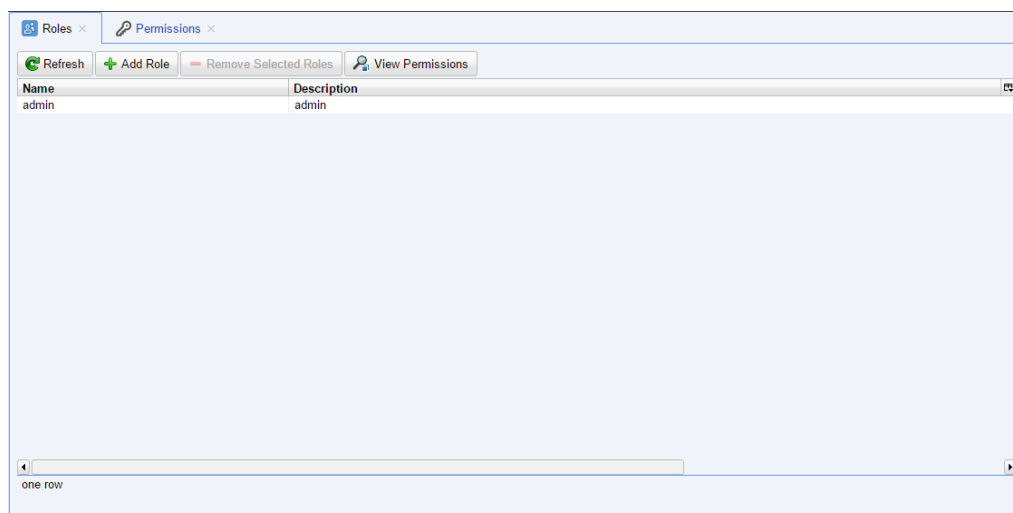
- ◆ Click the Grant button.

Alternatively, you can grant permissions to a role as described in Section 4.11.7.


- ◆ **Revoke:** Removes the permissions assigned to a role.

## 14.2 Managing Roles

Click the **Roles**  icon to display the **Roles** tab. The window shows the list of roles defined in the environment, and enables adding and removing roles and viewing permissions.



The **Roles** tab displays the following buttons:

- **Refresh:** Refreshes the role list.
- **Add Role:** Adds a new role to the database. Click the  button to display the following window:

Enter a role name in the **Role Name** field and a description in the **Role Description** field.

Then, click **Add**.

Alternatively, you can add a role as described in Section 0.

- **Remove Selected Roles:** Removes selected roles from the database. Alternatively, instead of this, you can remove a role as described in Section 0.
- **View Permissions:** Displays the list of permissions assigned to the selected role.

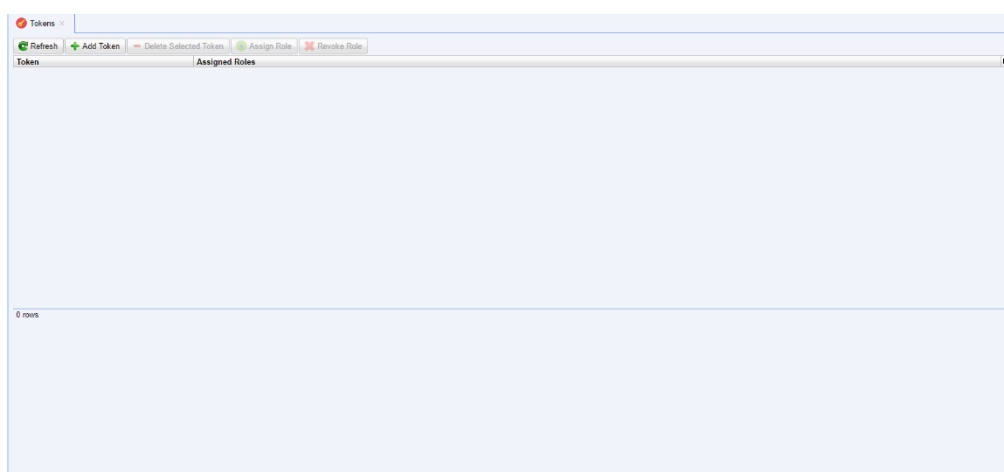
**Note:** Multiple roles can be selected using the Ctrl/Shift keys.

**Note:** Remove/grant activities are only enabled after selecting at least one role from the list of roles.

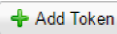
## 14.3 Managing Tokens

Click the **Tokens**  icon to manage tokens.

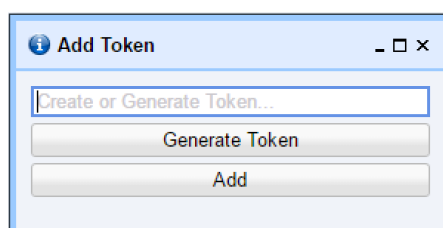
Tokens enable access to a web service. Permissions, as described on page 77, define which web services you can access. The window shows the list of tokens defined in the environment.



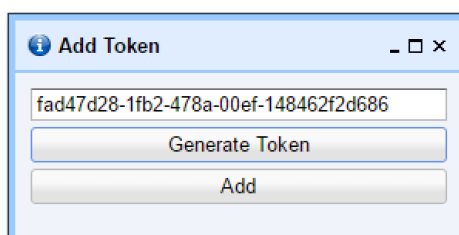
The **Tokens** tab displays the following buttons:


- **Refresh:** Refreshes the token list.
- **Add Token:** Adds a new token to the database. Click the  button to display the following window:

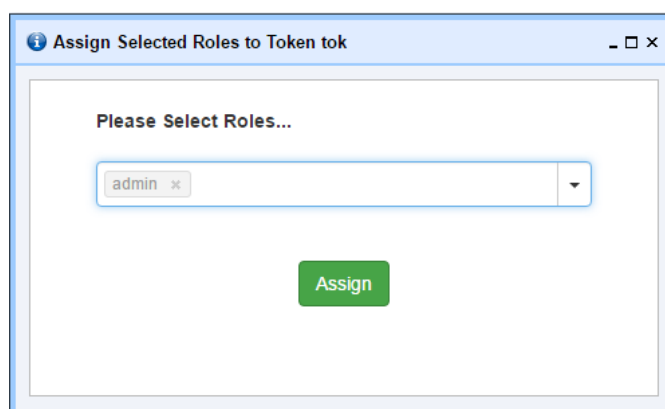




- a. Enter a token name in the empty field and click **Add**. Alternatively, you can add a token as described in Section 0.
- b. Click the **Generate Token** button in the *Add Token* window to generate a token with a random number, as shown below. Then, click **Add** to add the generated token.



- **Delete Selected Token:** Deletes the selected token and automatically removes its assigned assignment to all roles.
- **Assign Role:** Click the  **Assign Role** button to assign the token to a role. The following window appears:



Select the role to which to assign the token in the dropdown menu, and click **Assign**.

- **Revoke Role:** Removes the assignment of the token to a role.

When you activate web service, the required token must be supplied, as shown in the example below:

Test web services

Fill the following fields to test this web service

format  
☒ html ☐ json ☐ xml ☐ raw

methodName

token

name

Submit

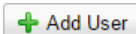
## 14.4 Managing Users

1. Click the **Users**  icon to manage users.

User	Super User	Assigned Roles
admin	true	admin

one row

The **Users** tab displays the following buttons:

- ◆ **Refresh:** Refreshes the user list.
- ◆ **Add User:** Adds a new user to the database. Click the  button to display the following window:

Add User

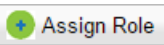
Username

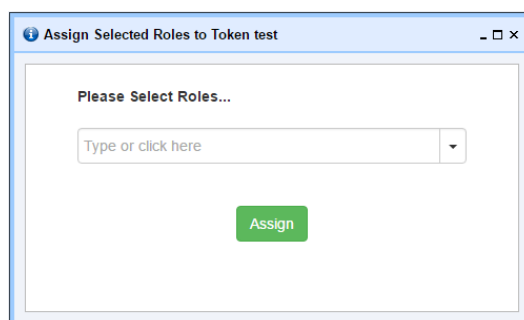
Password

☐ Super User

Add

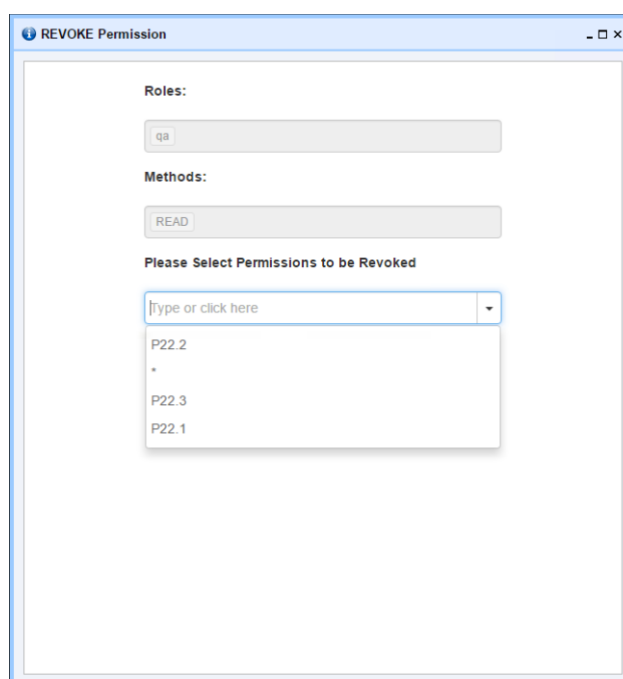
2. Enter a user name in the **Username** field, and a password in the **Password** field. Click **Add**. Check the **Super User** checkbox if you want the user to have additional permissions. Alternatively, you can add a user as described in Section 0.

- ◆ **Delete Selected User:** Deletes the selected user.
- ◆ **Assign Role:** Click the  button to assign the user a role. The following window displays:



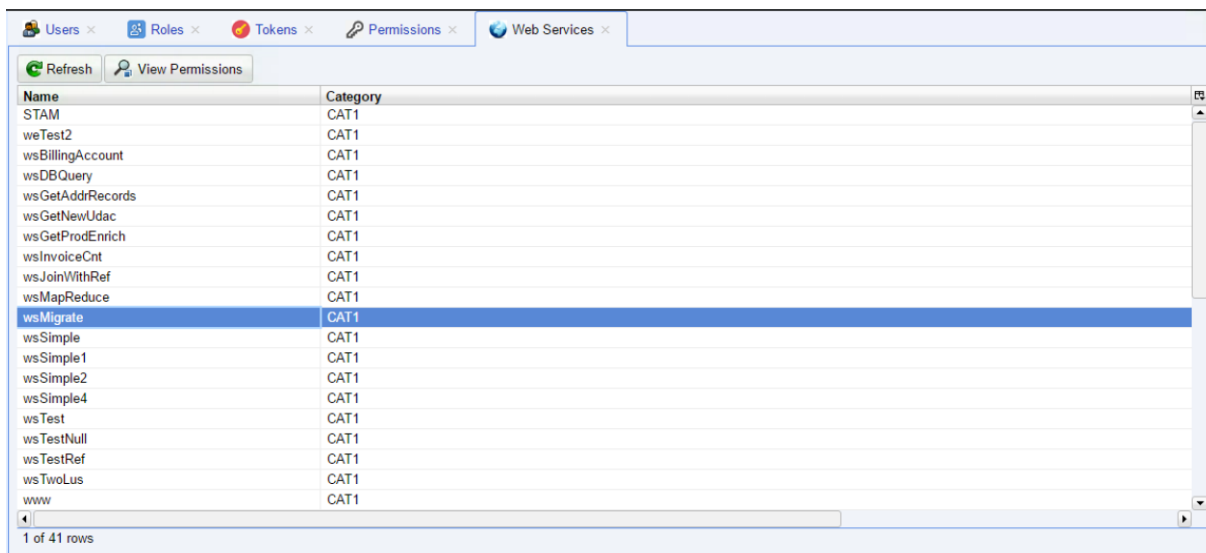
3. Select the role to assign to the user, and then click **Assign**.

- ◆ **Revoke a Role:** Removes the assignment of the role from the user, as shown below. To revoke all permissions at once, click the asterisk (\*) in the dropdown list.



## 14.5 Viewing Web Services

1. Click the **Web Services**  icon to view web services.




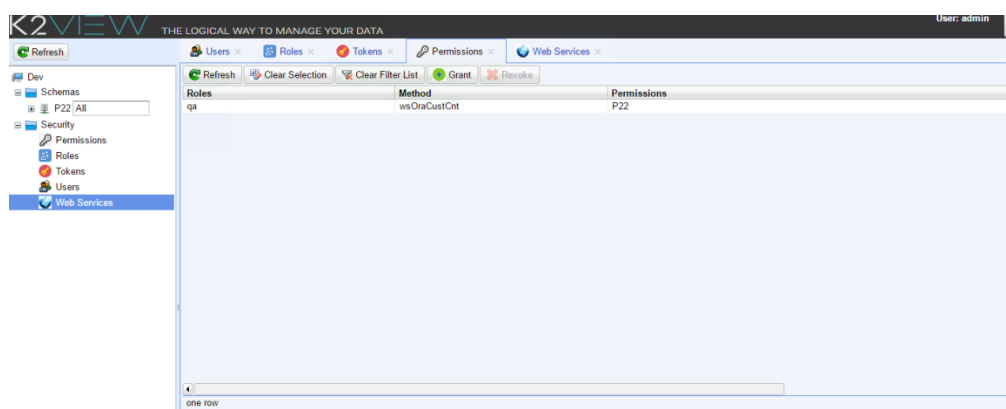
Name	Category
STAM	CAT1
weTest2	CAT1
wsBillingAccount	CAT1
wsDBQuery	CAT1
wsGetAddrRecords	CAT1
wsGetNewUdac	CAT1
wsGetProdEnrich	CAT1
wsInvoiceCnt	CAT1
wsJoinWithRef	CAT1
wsMapReduce	CAT1
wsMigrate	CAT1
wsSimple	CAT1
wsSimple1	CAT1
wsSimple2	CAT1
wsSimple4	CAT1
wsTest	CAT1
wsTestNull	CAT1
wsTestRef	CAT1
wsTwoLus	CAT1
www	CAT1


1 of 41 rows

The **Name** column specifies the name of the web service. The **Category** column indicates the category of the web service. The category organizes web services into groups, and can be any value set by the user.

The **Web Services** tab displays the following buttons:

- ◆ **Refresh:** Refreshes the web service list.
- ◆ **View Permissions:** Click the  **View Permissions** button to enable access to this web service for a specific role on a specific LU.



2. Click the  **Clear Filter List** button to clear any filters applied to the web services list.

## 14.6 Displaying Data

Fabric database data can be view by expanding the LU and References list, and navigating the Tables tree.

The data can be viewed for a specific instance. To specify an instance, enter the necessary information (for example, **11110**) in the text box next to the LU type name.

- **Note:** Data views correspond to user's permissions, so that a user only sees the instances to which he has the permission.

Selecting a specific table in the **Tables** list displays the contents of the table.

The screenshot shows the K2VIEW application interface. On the left, a tree view shows the database structure under 'Schemas' and 'Tables'. The 'Tables' list includes 'ALLERGIES', 'FAMILY\_HISTORY', 'INSTANCE\_TABLE\_CC', 'INSURANCE\_INFORM', 'INVOICE', 'K2\_TDM\_ROOT', 'LU\_PARAMS', 'M\_INVOICE', 'PATIENT', 'PAYMENT', 'S\_INVOICE', 'PATIENT\_MEDICATION', 'References', 'Security', and 'Permissions'. The 'M\_INVOICE' table is selected. The main pane displays a table with columns: PATIENT\_ID, INVOICE\_ID, ISSUED\_DATE, DUE\_DATE, STATUS, and BALANCE. The table contains 13 rows of data.

PATIENT_ID	INVOICE_ID	ISSUED_DATE	DUE_DATE	STATUS	BALANCE
1	1	2002-03-06	2016-08-03	OPEN	1992
1	2	2005-03-04	2016-09-04	OPEN	2879
1	3	2003-02-15	2016-01-11	OPEN	9343
1	4	2000-09-17	2016-10-17	CLOSED	15264
1	5	2001-05-23	2016-09-23	CLOSED	27689
1	6	2000-02-20	2015-12-06	CLOSED	17886
1	7	2012-02-17	2016-02-15	OPEN	29068
1	8	2006-04-27	2015-01-13	CLOSED	16028
1	9	2001-04-22	2016-07-30	OPEN	21297
1	10	2005-11-18	2016-04-29	OPEN	19750
1	11	2009-07-05	2016-08-27	OPEN	10597
1	12	2000-08-29	2016-11-03	OPEN	19988
1	13	2008-01-15	2016-08-07	CLOSED	22418
1	8888888			test	
1	1234564			test	

- Right-clicking the table's name enables the display of the following:

- ◆ **Table Properties:** Displays the table's attributes.

The screenshot shows the 'Table: ACCOUNT [0]' properties window. It displays table metadata including columns, data types, and index information. The table has 9 rows.

Name	Type	Index name	Index type
ACCOUNT_KEY	INTEGER		
FILE_KEY	INTEGER		
PROCESSOR_KEY	INTEGER		
ACCOUNT_NUMBER	TEXT		
DDA_NUMBER	INTEGER		
OPEN_DATE	TEXT		
EXPIRATION_DATE	TEXT		
ENTER_DATE	TEXT		
CHANGE_DATE	TEXT		

9 rows

Comment

- ◆ **SQL Query:** Executes any SQL query.

**K2VIEW** THE LOGICAL WAY TO MANAGE YOUR DATA

Refresh

PATIENT\_LU : ALLERGIES : \_DEV\_1 × PATIENT\_LU : PATIENT : \_DEV\_1 × PATIENT\_LU : M\_INVOICE : \_DEV\_1 × PATIENT\_LU : M\_INVOICE : \_DEV\_1 ×

Query

SELECT \* FROM M\_INVOICE LIMIT 100;

Execute Query Cancel Query

Column Chooser: Export To CSV export.csv

PATIENT_ID	INVOICE_ID	ISSUED_DATE	DUE_DATE	STATUS	BALANCE
1	1	2002-03-06	2016-08-03	OPEN	1992
1	2	2005-03-04	2016-09-04	OPEN	2879
1	3	2003-02-15	2016-01-11	OPEN	9343
1	4	2000-09-17	2016-10-17	CLOSED	15264
1	5	2001-05-23	2016-09-23	CLOSED	27689
1	6	2000-02-20	2015-12-06	CLOSED	17886
1	7	2012-02-17	2016-02-15	OPEN	29068
1	8	2006-04-27	2015-01-13	CLOSED	16028
1	9	2001-04-22	2016-07-30	OPEN	21297
1	10	2005-11-18	2016-04-29	OPEN	19750
1	11	2009-07-05	2016-08-27	OPEN	10597

- ◆ **Column Chooser:** Enables you to select specific columns to be viewed for the defined query (filters the results columns).
- ◆ **Export to CSV:** Saves the results in a \*.csv file format. The file can be downloaded to the default browser download location.
- ◆ **Data Filtering/Sorting:** Select the dropdown at the right of each column to sort in ascending/descending order and to filter according to the specified criteria. You can use the following operators: equal, not equal, less than, less than or equal to, greater than, greater than or equal to, null, not null and combinations.

CARD\_KEY FILE\_KEY PROC

2378695

248956488

12777

248956487

558956486

768956486

248956486

Sort Ascending

Sort Descending

Remove Sort

Show rows where:

less than

And

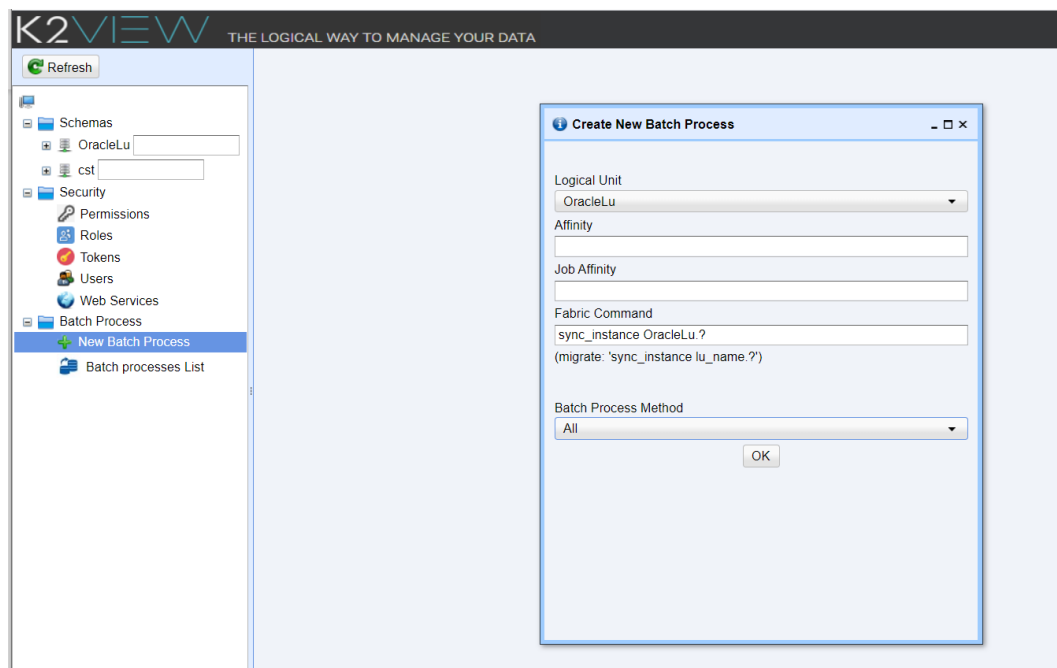
less than

Filter Clear

## 14.7 Running Batch Processes

Enables execution of batch processes, including migration commands, on Fabric through the Admin tool.

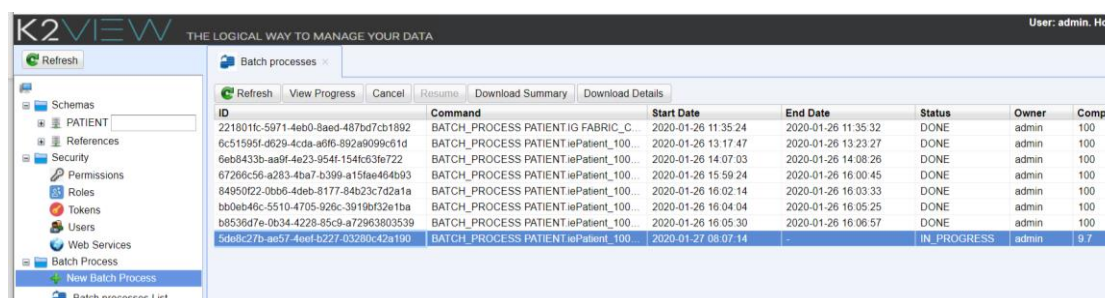
Double clicking the New Batch Process option opens the "Create New Batch Process" screen.



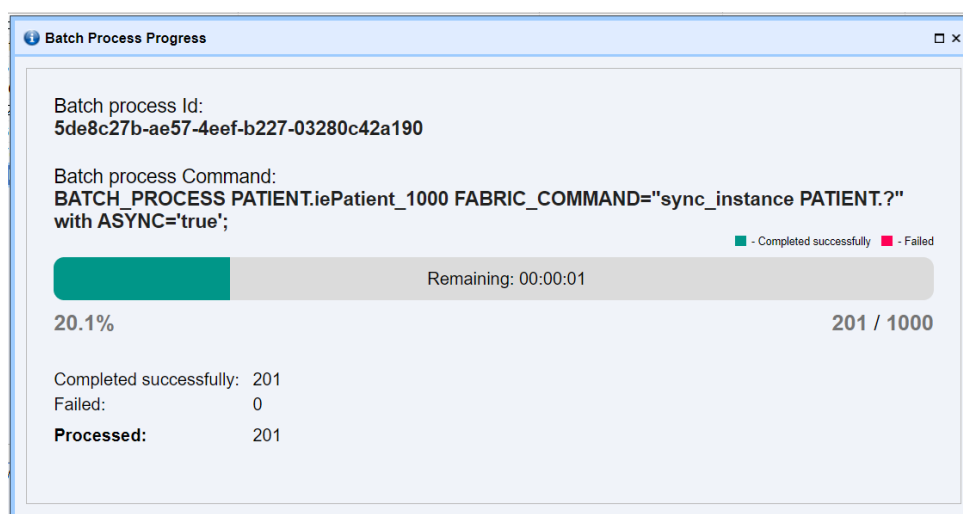
- **Logical Unit:** enables to select the LU to be executed.
- **Affinity:** An optional parameter for choosing list of nodes and DCs to be involved in the executed Fabric command.
- **Job Affinity:** Affinity for the migrate job.
- **Fabric Command:** You need to populate Fabric Command by “sync\_instance <LU Name>.?” to run a migrate process.
- **Batch Process Method** – enables to select the available methods:
  - ◆ All
  - ◆ By Instance Group
  - ◆ Based on SQL
  - ◆ Based on entities list- run batch process on a list of instances.

## 14.8 Viewing the Batch Process Executions

- Select the record that is in progress
- Click the View progress button.



The Batch Process Progress screen appears and displays the progress status of execution using the progress bar.



- Cancel** – enables cancelling the batch process at the middle of the process.
- Resume** – enables resuming the batch process from the point it was stopped/cancelled. It also processes all the failed entities.

**Note:** Once the batch process is completed, there is no option to open the progress screen.



- Viewing the migration reports:
  - a. Download Batch Process Summary

#### Batch process summary report for batch process 5de8c27b-ae57-4eef-b227-03280c42a190

##### Batch process details:

Command: BATCH\_PROCESS PATIENT.iePatient\_1000 FABRIC\_COMMAND="sync\_instance PATIENT.?" with ASYNC="true";

Status: DONE

Executed by: admin

Logical unit: PATIENT

Nodes list: 935c35c8-dd8b-4f55-83b6-185cc49e2cef

Level	Name	Status	Start time	End time	Duration	Remaining dur.	Remaining	Total	Succeeded	Failed	UNCHANGED	ADDED	UPDATED	% Completed	Errors (p
Node	935c35c8-dd8b-4f55-83b6-185cc49e2cef		2020-01-27 08:07:14	2020-01-27 08:08:47	00:01:33	00:00:00	0	--	1000	0	0	0	1000	100	12
DC	DC1		2020-01-27 08:07:14	2020-01-27 08:08:47	00:01:33	00:00:00	0	--	1000	0	0	0	1000	100	12
Cluster	--	DONE	2020-01-27 08:07:14	2020-01-27 08:08:47	00:01:33	00:00:00	0	1000	1000	0	0	0	1000	100	12

- b. Download Batch Process Details

#### Batch process details report for batch process 5de8c27b-ae57-4eef-b227-03280c42a190

##### Batch process details:

Command: BATCH\_PROCESS PATIENT.iePatient\_1000 FABRIC\_COMMAND="sync\_instance PATIENT.?" with ASYNC="true";

Status: DONE

Executed by: admin

Logical unit: PATIENT

Nodes list: 935c35c8-dd8b-4f55-83b6-185cc49e2cef

Entity ID	Node id	Status	Error	Results
_dev_324	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_329	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_142	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_255	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_401	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_351	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_26	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_66	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_619	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_506	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}
_dev_605	935c35c8-dd8b-4f55-83b6-185cc49e2cef	COMPLETED		{"UNCHANGED":0,"ADDED":0,"UPDATED":1}

---

## 15 Customer Support

Please contact our offices for further information on K2View products.

K2View email: [support@k2view.com](mailto:support@k2view.com)