K2VIEW FABRIC STUDIO 6.1

USER GUIDE

April, 2020

Confidentiality

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

This document and information contained herein are delivered 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 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, any document and/or copyright contained herein may not be reproduced by any means.

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.

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

1 Introduction

1.1 About this User Guide

This document details the use of K2View Fabric Studio.

K2View Fabric studio is used to define and manage Fabric projects.

1.2 K2View Fabric Technical/Conceptual Overview

In order to provide full data synchronization capabilities, K2View Fabric uses a game-changing data model to retrieve and store data: the Logical Unit.

Most database management systems organize the data storage based on the type of data being stored (e.g. customer data, financial data, address data, device data). This model translates into a large number of tables that must be queried using complex joins every time one wants to access business relevant data (e.g. how many payments has this customer made within the past three months?).

Fabric looks at data from a different perspective. K2View's Fabric retrieves and stores the data based on business logic, hence the name Logical Unit. This allows businesses to impose the K2View Fabric base schema based on their needs, as opposed to trying to fit them into a pre-defined structure.

In K2View Fabric every business-related object (e.g. Customer, Merchant) is represented by a Logical Unit Type (commonly referred to as an LU throughout this guide). Each Logical Unit Type is then associated with a schema. Within the Schema you define relevant data structures associated with one Logical Unit Type. This process is either automated, using Fabric's Auto-Discovery module (see Section 17.3), or performed manually. The result is a business-oriented structure incorporating data from tables and objects from as many systems as needed.

Using the Schema, K2View Fabric automatically retrieves all information associated with a specific Logical Unit Type using embedded, in-memory migration capabilities.

The data is then accessed, distributed, and stored in K2View Fabric as Logical Unit Instances, allowing for incredible performance, enhanced security, high availability, and customizable data synchronization.

As such, the Logical Unit concept is a bridge between scattered, hard to maintain data and highly available, business-oriented data.

1.3 K2View Fabric - Studio & Run Time

K2View Fabric is based on a tiered application-server architecture, where the user interface is a Windows-based application. The server side is a cloud-based platform agnostic solution.

K2View Fabric consists of two distinct layers:

- **Implementation Layer**: The design, development, and deployment are performed in this environment.
- **Execution Layer**: This layer is defined on a Linux server and the actual execution of the project is performed in this environment.

1.4 Terminology

The following is a summary of the key terminology used in this guide.

- **Database Types**

  Type of database to be used by the interface.

- **Interface:**

  A Database connection used by **K2View Fabric Studio** that connects projects' objects to an external source database.

- **Logical Unit Type/Schema:**

  The Logical Unit is one of the K2View Fabric's foundations. The Logical Unit is a hierarchical presentation of all the business entities related to a root table. This is a relational schema of tables.

- **K2VIEW Fabric Object:**

  Any object, defined in K2View Fabric Studio such as a table, function, translation, and so on.

- **Table:**

  An object, which contains data: either directly from one or more source tables, or as calculated data.

- **Instance ID:**

  Instance ID is the ID of a single, specific business object. A Logical Unit type (LUT) can contain many instance IDs. For example, a customer ID 12345 may represent a specific customer of an instance ID in a CUSTOMER Logical Unit type (schema).

- **Instance Group:**

  A list of instance IDs to be processed.

- **Shared Objects:**

  Objects shared by all Logical Units of a Project.

- **Globals**:

  Variables accessible to every object within the project implementation.

- **Transformation:**

  The process of mapping input data to output data, typically the legacy information into K2View Fabric. During this process, data can be modified, manipulated, or cleansed based upon business needs or rules.

- **SVN:**

  Sub-version is a software configuration control, maintaining current and historical versions of the K2View Fabric objects, allowing multiple users to work simultaneously and ensuring that their work is synchronized.

- **Sync:**

  K2View Fabric flexible data synchronization features are driven by its Smart Data Controller. Any time data is accessed in K2View Fabric, the Smart Data Controller compares the current state of the data in K2View Fabric to the synchronization parameters and updates the data if needed. This is done regardless of whether it involves a change in the K2View Fabric schema, or was triggered by one of the synchronization modes (depended by the sync method), which are described in the *Introduction to K2View* white paper.

  Sync is supported in 3 levels, with inheritance capabilities between them: LU Schema, Table, and Population.

1.5 All-In-One

K2View Fabric features a set of embedded data management functions to populate, store, and present data:

- Embedded ETL (Extract-Transform-Load) capabilities- performing complex transformation logic on data from source systems.
- Embedded data masking – masking the source data before its migration into the K2View Fabric Data Base.
- Flexible data synchronization.

These data management features are the key differentiators between K2View Fabric and other data management systems, including modern distributed systems. For these traditional solutions, data management is cumbersome, risky, and expensive. In K2View Fabric, data management is part of the database capabilities. It allows data to be retrieved, validated, and enriched automatically, without using any transformation scripts or third-party tools.

2 Understanding the K2View Fabric Workflow

2.1 Workflow Overview

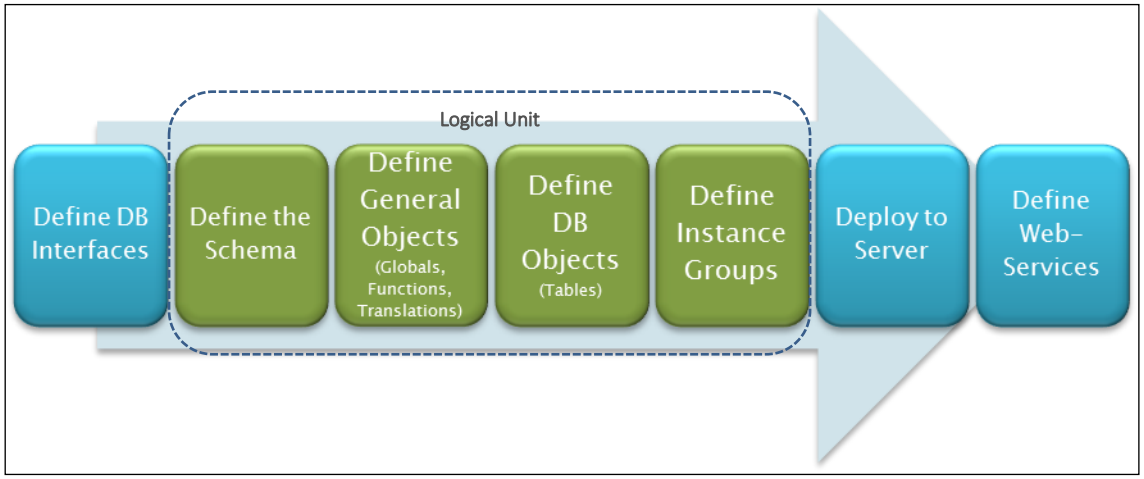
The following diagram presents a typical workflow for creating a K2View Fabric project.

Step 1: Define DB interfaces to connect to your legacy systems.

Step 2: Define the Logical Unit and its Objects. This includes defining data mapping and transformations used for the Logical Unit, and the subset of data records to use.

Step 3: Deploy definitions to K2View Fabric.

Step 4: Define and Deploy Web-Services.



Note: The order of the sub-steps in defining the Logical Unit (Step 2) is not absolute. It is possible, for example, to define tables, translations, and then more tables and functions.

2.2 Workflow - Procedures Map

You can find the procedures performed at each stage of the workflow in the following index:

Workflow Step	Procedure	Section in User Guide
Define Database Type	Create a new Database Type	55.2, 5.3
Define DB Interface	Create Data Source Interface	7.3
	Edit Interface Settings	7.5
Define Logical Unit DB	Define LU	8.3
Define General Objects	Define Translation	9.3.2
	Creating Functions, Adding Parameters, Editing Function	9.2.2, 9.2.3
	Editing Globals	09.4.1
Define DB Objects	Adding Tables to Logical Unit	7.7
	Linking Table to Hierarchy	7.7.2
	Adding Functions	8.7.3
	Adding Translation	8.7.4
	Adding Global	8.7.5
Define Instance Groups	Defining Instance Groups	7.15
Deploy to K2View Fabric Server	Deploying to Server	7.13
Define Web Services	Creating Web Services	15.1

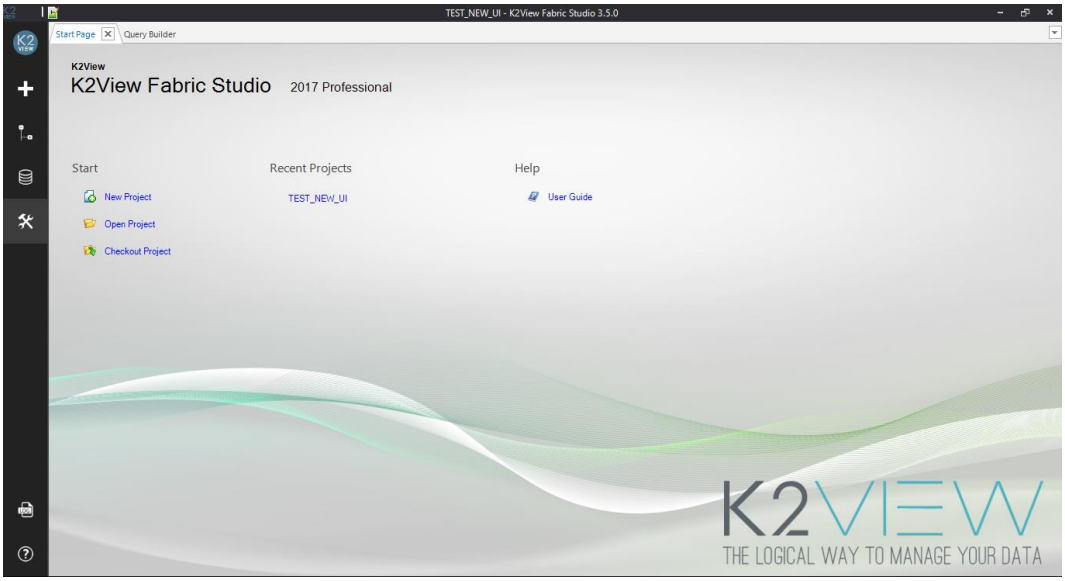
3 K2View Fabric Studio User Interface

K2View Fabric Studio is designed as an integrated tool that allows you to implement mapping rules into K2View Fabric, debug, and view mapped data.

The design of the main working space is similar to the standard look-and-feel of Windows based development environments.

K2View Fabric Studio provides integration with a Configuration Control system based on the Apache Sub-version (SVN) Standard. More information on this integration and using the SVN system is provided in Chapter 4.

3.1 Start Page



The Start page includes links to several project activities.

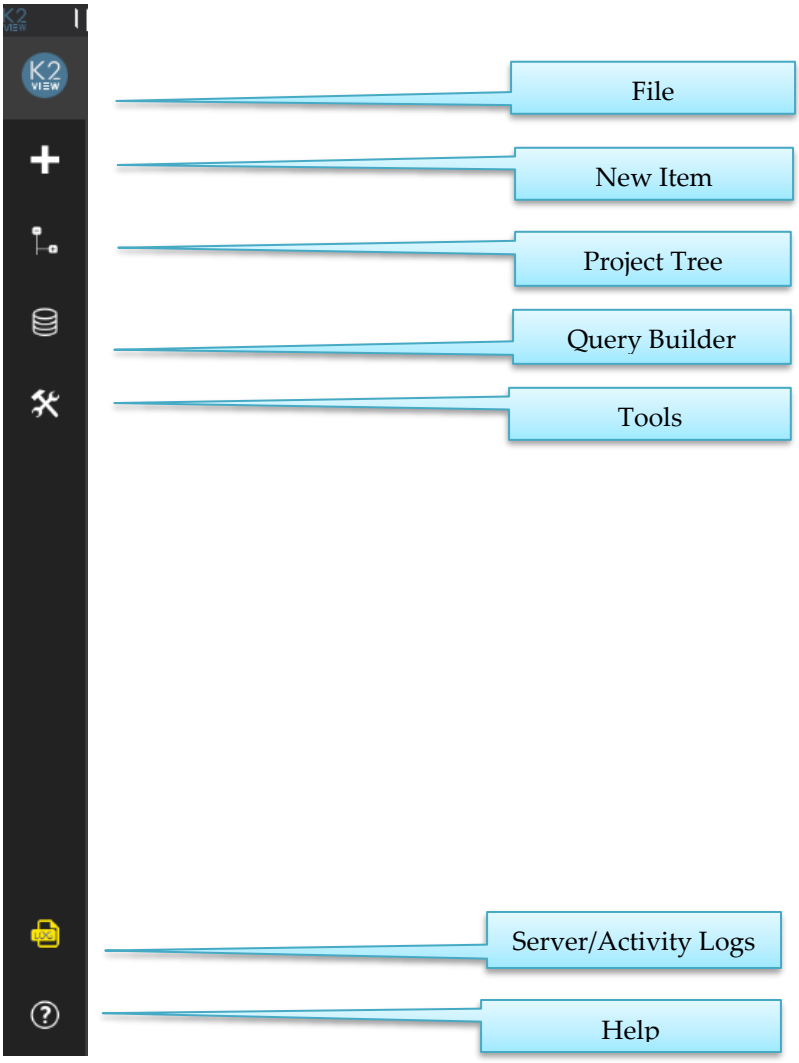
The Start Page provides links to start working on a K2View Fabric project. In particular:

- Start a new project, with or without configuration control.
- Open an existing project.
- Checkout a project from the configuration control.
- Quick links to recently opened projects.

In addition, the Start Page provides links to Help information including a full User Guide.

3.2 Fabric Studio- Left Taskbar

Fabric Left Menu has the following icons:



3.2.1 File Tab

File tab displays the following options:

- **New Project**- create new project
- **Open Project**- open existing project
- **Checkout Project**- checkout project from SVN or Git repository
- **Close Project**
- **Recent Projects**

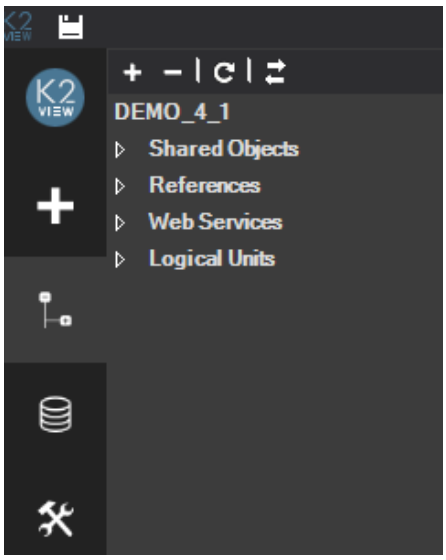
3.2.2 New Item Tab

The New Item (+) tab (Ctrl+N) opens a new item window in the selected LU or shared objects. You can select one of your LUs from the current Logical Unit dropdown list to create the new item under the selected LU.

To create a new item under the shared objects, you need to set the LU of the current Logical Unit dropdown list with an empty value.

3.2.3 Project Tree Tab

Project tree tab (CTRL+T) opens the project tree:



The Project Tree displays the components of the current Project as a hierarchy of entities. The main elements of the hierarchy include:

- Project - this is the top-level of the hierarchy. Each Project has five children:
  - **Shared Objects** – objects that can be used by any Logical Unit within the project.
  - **References** – reference information that may be used across instance (see Section 14).

- **Web Services** – collection of all functions that are defined as web services for this project (see Section 15).
- **Logical Units** – collection of all Logical Units defined in the project (see Section 7).

3.2.3.1 Shared Objects Tree

- The Shared Objects can be used by all Logical Units and references of the project and may include a sub-hierarchy that includes the following:
  - **Globals** – a set of global parameters defined for this scope (see Section 9.4).
  - **Environments** (see Section 14).
  - **Database types** – type of databases to be used when defining interfaces.
  - **Interfaces**- a list of source interfaces defined for the project (see Section 5).
  - **Functions** – Sets of functions, which can be used for transformation of the data (see Section 9.2).
  - **Java**- Java files and resource files.
  - **Translations** - sets of translation, which can be used for transformation of the data (see Section 9.3).

3.2.3.2 Reference Tree

- The Reference tree may include a sub-hierarchy that includes the following:
  - **Globals** – a set of global parameters defined for this scope (see Section 9.4).
  - **Functions** – sets of functions, which can be used for transformation of the data (see Section 9.2).
  - **Java**- Java files and resource files.
  - **Translations** - sets of translation, which can be used for transformation of the data (see Section 9.3).
  - **Tables** - lists the data tables configured for this scope.

3.2.3.3 Logical Units Tree

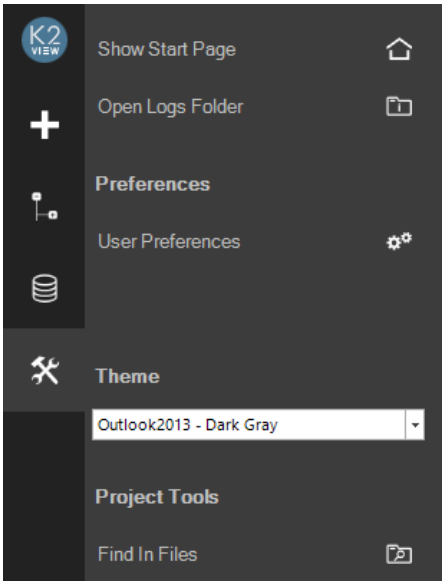
- The Logical Units tree includes the list of Logical Units, defined in the project. Each Logical Unit may include a sub-hierarchy that includes the following:
  - **Schema** – hierarchy of database tables (see Section 7.6.1).
  - **Globals** – a set of global parameters defined for this scope (see Section 9.4).
  - **Functions** – Sets of functions, which can be used for transformation of the data (see Section 9.2).
  - **Java**- Java files and resource files.
  - **Translations** - Sets of translation, which can be used for transformation of the data (see Section 9.3).
  - **Tables** - lists the data tables configured for this scope.
  - **Parsers** (see Section 13).

- **Instance Groups** – (only Logical Units) lists the groups of source data to be retrieved and loaded into the K2View Fabric (see Section 0).

> **Note:** All operations performed by the project tree options are accessible from the context menus for different entities in the Project Tree. For example, creating a new function is accessible from the context menu when right clicking the Functions object of the Project Tree.

3.2.4 Tools Tab

When a project is opened, the Find in Files and Project Tools buttons will be available for selection.



3.2.5 Help Tab

User Guide- opens this document for browsing.

> **Note:** You can press the F1 key to open the User Guide document.

About – displays the version of K2View Fabric Studio and license information.

3.3 Working Area

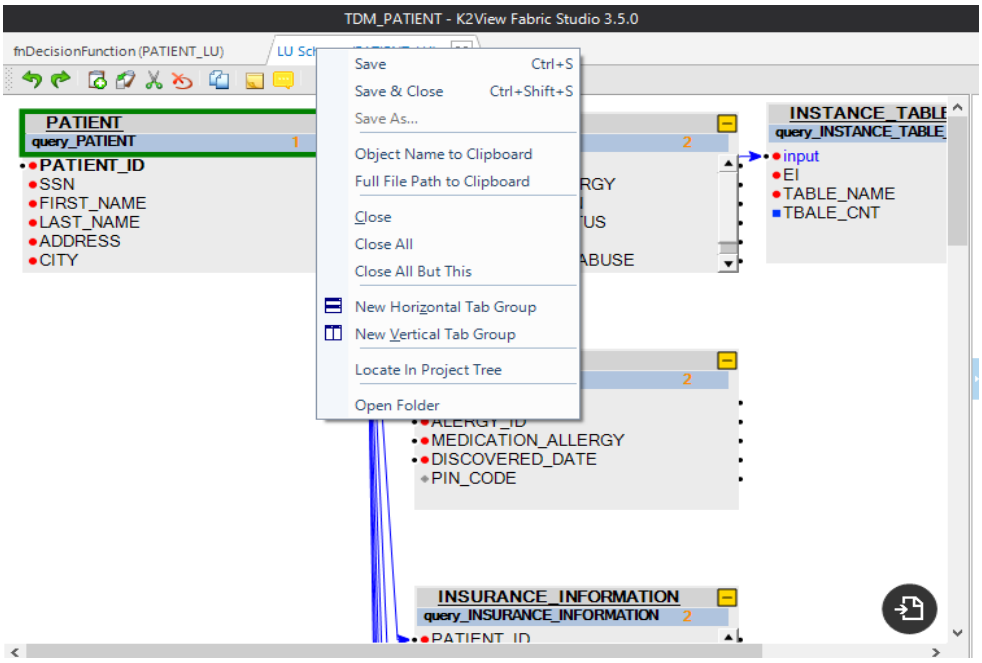
The main area of the K2View Fabric Studio window is occupied by panes that are specific to the operations that a user requires.

Action	What is Displayed	Reference Topic
Displaying the Logical Unit Schema	Logical Unit Schema window	Section 7.6.1
Creating new function	Function Manager window	Section 9.2.2
Creating new translation	Translation Manager window	Section 9.3.2
Creating new global	Globals Manager window	Section 9.4
Editing the data mapping and transformations of different database entities	Table Mapping window	Section 8
Viewing Data	Data Viewer window	Section 7.12
Creating new specific data retrieval procedures	Query Builder window	Section 17.2.1
Create and edit a parser	Parser Map window	Section 13
Create and deploy environments	Environments window	Section 14

> **Note:** When multiple functional windows are open in the working area, you can switch between different windows using the tabs that appear along the top of the main working area pane

3.4 Window Tab Context Menu

When you right click the window tab header, a context menu is displayed. The following commands are supported:



Command	Description
Save	Save object
Save & Close	Save object and close it
Save As	Create a new instance of the object with a different name
Object Name to Clipboard	Copy the object name into your clipboard. For example: fnEnrichmentLuParams
Object Name to Clipboard	Copy the object name including its file path into your clipboard. For example: C:\Users\Dell\Documents\K2View Fabric Studio\Projects\TDM_PATIENT\Implementation\LogicalUnits\PATIENT_LU\Functions\LuParams\fnEnrichmentLuParams.k2function.xml
Close	Close object
Close All	Close all open objects
Close All But This	Close all open objects except the one that you are working on
New Horisontal Tab Group	Set horisontal view for your open objects
New Vertical Tab Group	Set vertical view for your open objects

Command	Description
Locate in Project Tree	Select the open object on the objects tree
Open Folder	Open the folder with your object

3.5 User Preferences

This section describes the preferences that you can define. The preferences are grouped according to the following topics:

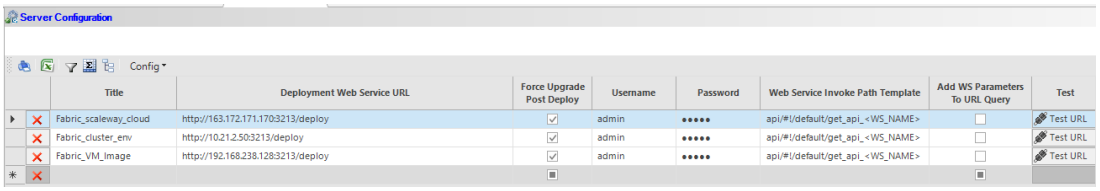
- General Preferences
- Server Configuration
- Application Shortcuts
- SVN Preferences
- Git Preferences

3.5.1 General Preferences

This section includes settings for:

- **Default Projects Directory**, a folder on the local system where the Project is saved. This field is editable.

3.5.2 Server Configuration



The Server Configuration option allows you to define the URLs for the K2View.

Fabric database servers. For each server you supply:

- **Title** – The K2View Fabric server name.
- **URL** – Displays a generic URL for the deployment. You will supply the hostname (or host IP address) in place of the "<host>" field.
  - http://<host>:3213/deploy

> **Note:** Do **not** change any other part of the URL.

- **Username** – Username used for the database access authentication.
- **Password** - Password used for the database access authentication.

- **Force Upgrade Post Deploy** – Performs Sync force after deployment. If the option is unchecked- performs Sync only in case of schema changes.

> **Note:** Your LU is always synced when you deploy it, if you select 'Force Upgrade Post Deploy' regardless the sync method of your LU. If you set a decision function as a sync method, your decision function is not executed during the deployment, when 'Force Upgrade Post Deploy' is selected.

- **Web service invoke path template- By** default, the web services invoked using the Swagger, but you can change the invoke path template.
- **Test** – Use this button to validate the URL.

> **Note:** You can define more than one URL. These URLs appear in the list when using the 'Deploy to Server' option (Section 7.13).

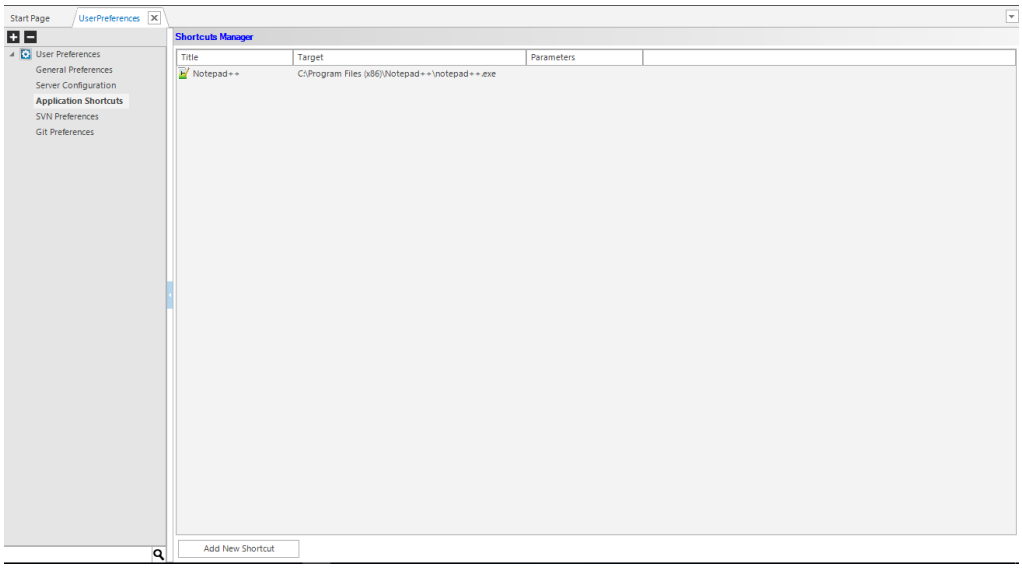
3.5.3 Application Shortcuts

The Application Shortcuts option in the User Preferences tree allows users to define shortcuts to use other applications from within K2View Fabric Studio, without exiting the application.

To add a new application shortcut

(1) Click the **Application Shortcuts** item in the User Preferences tree (left of the window pane).

Shortcuts Manager Pane is displayed



(2) Click the **Add New Shortcut** button (at the bottom of the pane).

(3) Enter the application name (for display purposes) in the **Title** field.

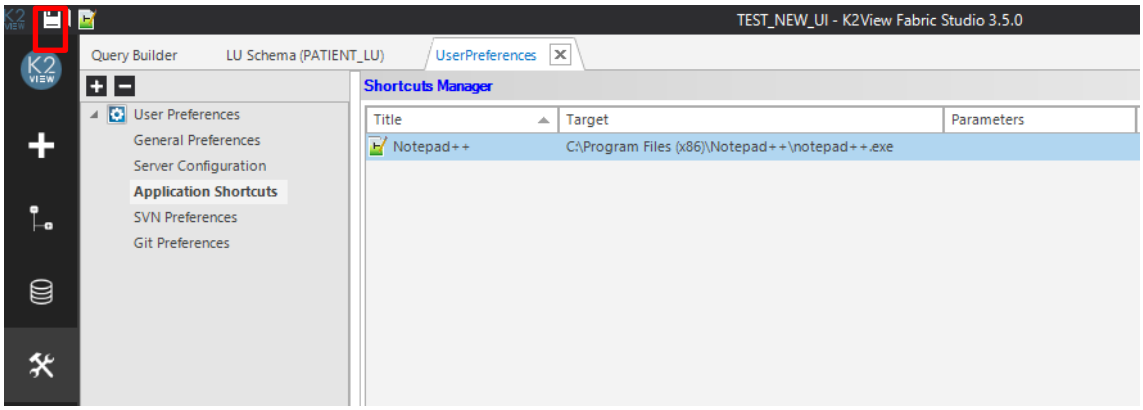
(4) Enter the path to the application file in the **Target Path** field.

(5) Enter any parameters for the application in the **Parameters** field.

(6) Click **OK** to finalize the shortcut definition.

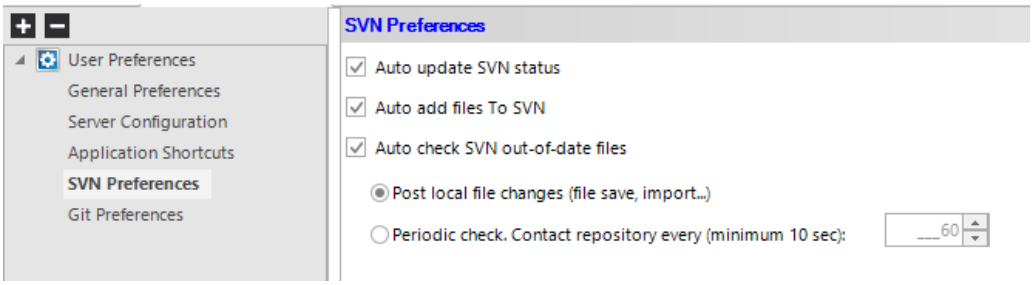
(7) Click the **Save** icon ( ) on the top left corner of your window or use **CTRL+S** to save the updated shortcuts.

The new shortcut appears in the upper area of the window.



3.5.4 SVN Preferences

The following SVN parameters can be set from the **SVN Preferences** item in the User Preferences tree:



- **Auto-update SVN status** - Check this box to see the SVN status indicator icon for each of the components in the project tree whenever a component's status was changed and saved.
- **Auto Check SVN last revision** –K2View Fabric Studio checks this box each time an open window is saved or closed. A message appears indicating that there is a newer revision of the project in the SVN repository and, therefore, an update is required.
- **Auto add files to SVN: adds** ability to disable adding files automatically to the SVN in the user preferences window.

3.5.5 GIT Support

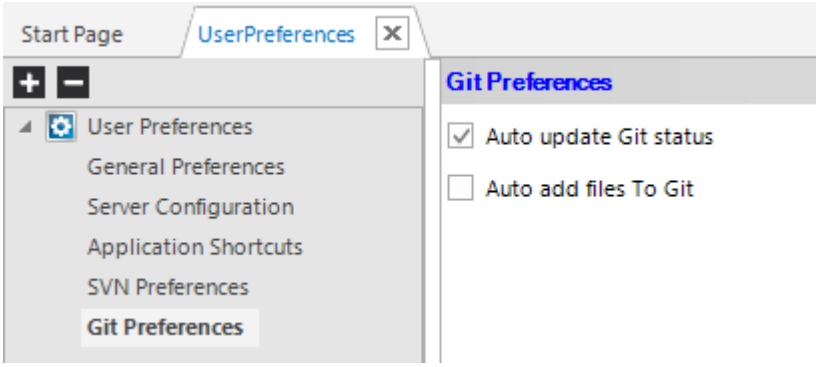
- Supports creating new project with GIT.
- Supports checkout project from GIT.
- All other GIT commands are enabled only if TortoiseGit is installed on a local machine. These commands are under a new TortoiseGit context menu in the project tree.

> **Note:** In GIT, the studio does not add project files automatically to GIT (user can turn this feature on in the GIT user preferences or add them using the TortoiseGit Add command).

3.5.6 GIT Preferences

GIT preferences under UserPreferences tab:

- Allow the user to disable getting files status automatically.
- Allow to enable/disable adding files automatically to GIT

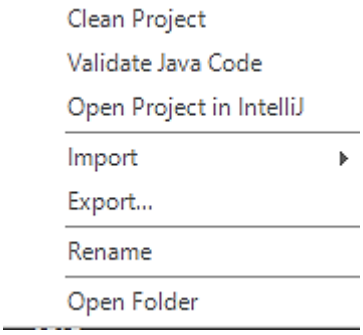


3.6 Project Tree Context Menus

The Project Tree pane, on the left side of the K2View Fabric Studio window, displays the project entities. Each of the entities is a starting point for different actions, accessible through a context menu. Right-click the object to display its context menu. The context menu displays options to start actions relevant to the particular object.

3.6.1 Folder Level Common Options

The following activity options are common for all Project folders:



- **Item Properties** enables you to add balloon comments to the item. The comments are displayed when floating over the item. These are helpful in the large scale projects when item names may not clearly define the item's purpose.
- **Export** enables you to export an object from the project (entire project, project folder, or a single project object) to a file (see Section 4.7). This may be used for backup purposes or moving project folders or objects between different projects.
- **SVN** is the menu for the SVN Options.
- **Refresh SVN status** enables users to synchronize all local items with the SVN.

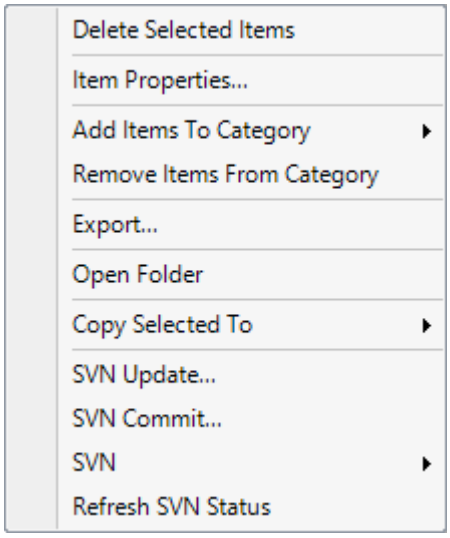
> **Note:** The SVN options only appear for projects that are using the Configuration Control (see Section 4.1) of K2View Fabric Studio.

- **Open Folder** enables you to open a specified folder in Windows Explorer.
- **Validate Java Code** enables you to compile Java code and check for errors
- **Open Project in IntelliJ** enables you to open the project in IntelliJ IDE.

3.6.2 Object Level Common Options

The following activity options are common for all object level items, for example, a specific function or table in the Project Tree:

- **Delete Selected Items**: enables you to delete the selected entities.
- **Item Properties**: enables you to add balloon comments to the item. The comments are displayed when floating over the item. These are helpful in a large scale project when item names may not clearly define the item's purpose.



- **Add Items To Category**: enables you to add a "sub-folder" of entities with a common category.
- **Export**: enables you to export an object from the project (entire project, project folder, or single project object) to a file (see Section 4.7). This may be used for backup purposes or moving project folders or objects between different projects.
- **Open Folder**: enables you to open the object folder in Windows Explorer.
- **Copy Selected To:** enables you to copy an object to Shared Objects or a different Logical Unit.
- **SVN Update:** Since a typical user works on a local machine, the best practice is to synchronize the local copy with the central repository. The Update option enables users to perform manual synchronization at any time. This option enables users to choose whether they want to update to the HEAD Revision or to an earlier revision.

> **Note**: The recommended revision is the HEAD Revision, unless there is a specific reason to update to a specific revision.

SVN Commit: commits the latest object changes to the main repository.

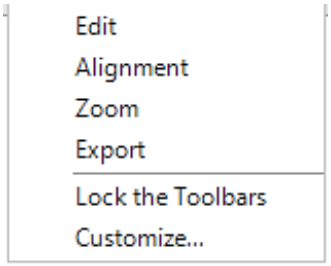
SVN: the menu for the SVN additional Options.

Refresh SVN Status: enables users to synchronize all local items with the SVN.

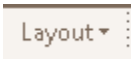
> **Note**: The SVN options only appear for projects that are using the Configuration Control (see Section 4.1) of K2View Fabric Studio. For more information on these options see Section 4.3.1

3.7 Diagrams Toolbars

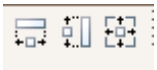
There are several toolbars that are available for diagrams windows- Logical Unit Schema, Population, Parser. You can customize your diagram window by right clicking the upper toolbar, and then selecting one or more of the following options:



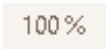
- Standard editing toolbar that includes:

	Undo last change.
	Redo last change.
	Copy selected item to clipboard.
	Paste item from clipboard.
	Cut selected item and copy to clipboard.
	Delete selected item.
	Select all.
	Add note to an object.
	Add balloon note.
	Layout menu - **Layered layout** or **Tree layout** options.

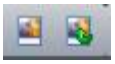
- Alignment toolbar:

	Left/middle/right alignment
	Top/center/bottom alignment
	Size alignment

- Zoom toolbar that includes:

	Current zoom factor
	Zoom out
	Zoom In
	Cancel Zoom
	Fit to Page
	Focus on selection

Export toolbar that includes:

	Print Preview/Print
	Export To Bitmap/Export to SVG

Customized- you can add or remove toolbars, or you can customize selected toolbars in your window.

> **Note:** Your customized toolbar will be available for the selected diagram window even if you close and then reopen your diagram for your current session.

3.7.1 Edit Toolbars Icon

Each diagram window has the following icon in the bottom right corner: 

When you click on this icon, the following options are displayed:

- Undo icon
- Redo icon
- Delete icon
- Edit toolbar
- Zoom toolbar
- Alignment toolbar
- Print & Export toolbar

4 Working with Projects

A project in K2View Fabric is a consolidation of rules for the purpose of transforming data from one or more source systems or databases into the K2View Fabric data structure.

4.1 Validate the Java Code When a Project Is Opened

The Fabric Studio compiles all Java code in your project when you open the project. The compilation errors, if exist, are displayed in the compilation errors tab.

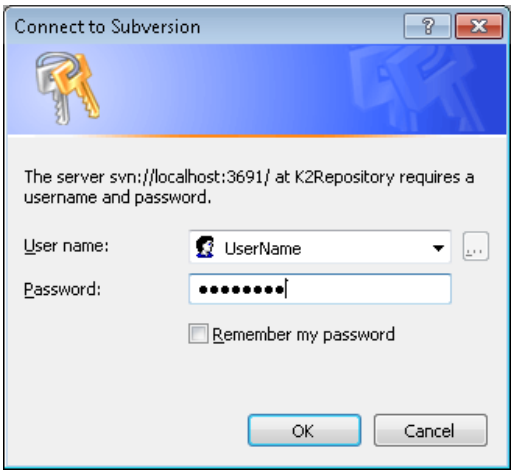
4.2 Configuration Control

K2View Fabric Studio has a built-in Configuration Control functionality based on the well-known Apache Sub-version (SVN) Standard or GIT Standard. Configuration control maintains current and historical versions of files. This enables cooperative and simultaneous work by multiple users while ensuring the integrity of their work. Using this feature allows your team to revert to an older version of the project files if errors are found in the current version.

The Configuration Control Manager is responsible for the configuration of the SVN for a project, as well as the repository location, attributes, and permissions. Configuration of these settings is performed by the Administrator with the K2Admin tool as described in the K2Admin User Guide.

4.2.1 SVN- Logging into the Configuration Control Project

When a user opens a project under Configuration Control, the Configuration Control login window of the most recently opened project is displayed.



☞ **To login to the Configuration Control system**

Enter the SVN username in the **User name** field.

(1) Enter the SVN password in the **Password** field.

(2) Click **OK**. (0)

*The K2View Fabric Studio **Main Window** (Section 4.3) is displayed.*

4.2.2 Fabric Studio GIT Setup

4.2.2.1 GIT Server

(1) Make sure you have GIT Server available (You can also use GitHub).

(2) Create new Repository on your GIT server for your New Project.

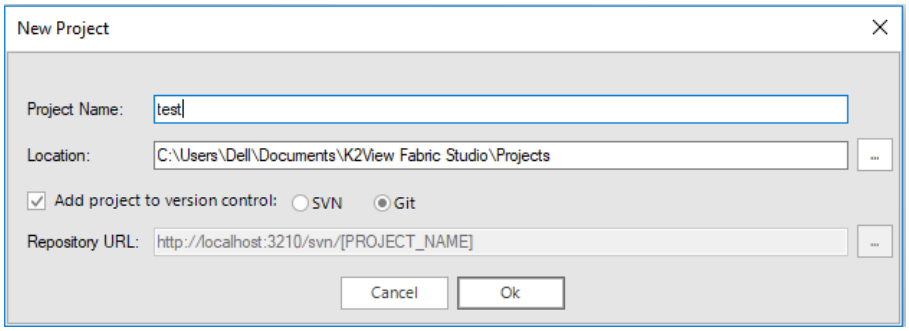
(3) Copy Repo URL and create relevant permissions for users.

4.2.2.2 GIT Client

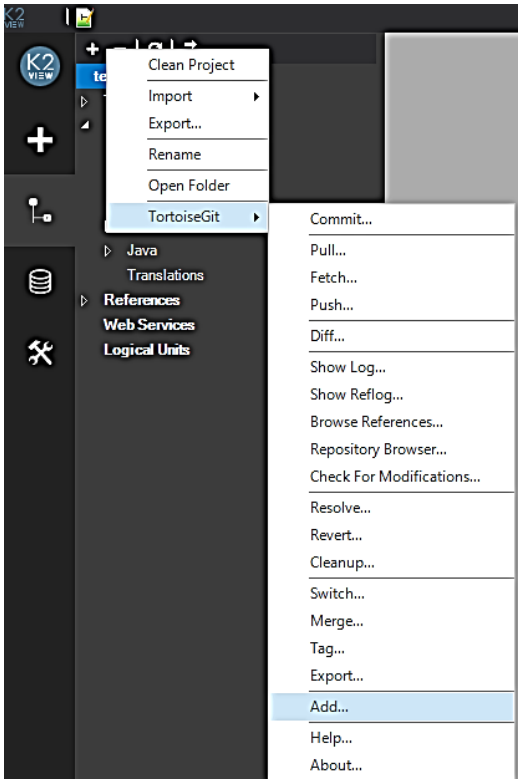
Download and install TortoiseGit to your Studio window's Machine.

4.2.2.3 Setting up GIT on Fabric Studio

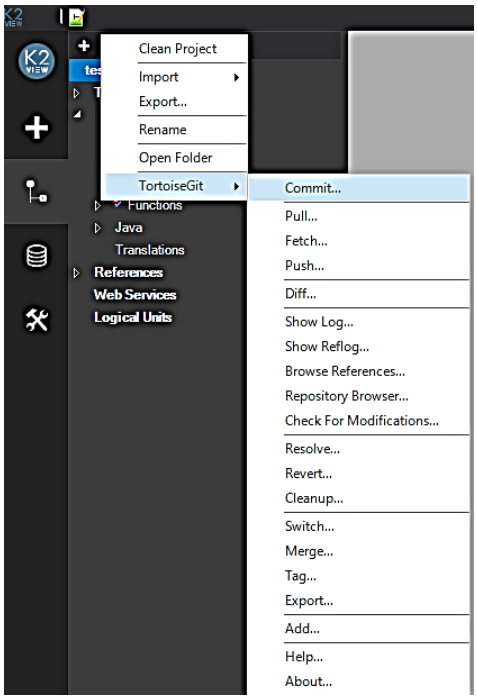
In Fabric Studio: Create New Project and add it to GIT:



(1) Add your new project Items to GIT:



(2) Commit your new project Items:



(3) Right-click the TortoiseGit option and select "Push" to push your project to the GitHub Repository.

TortoiseGit window is opened.

(4) Click "Manage" and populate your repository URL.

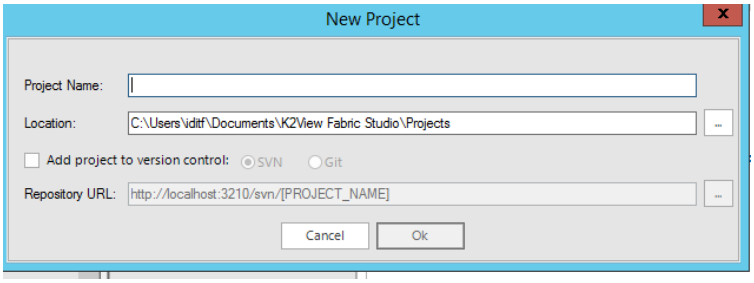
On your first "Push", GIT will ask for User and Password.

4.3 Opening a New Project

☞ To create a new project

(1) Click **New Project** on the Start Page (Section3.1).

The New Project dialog box is displayed.



(2) Enter the name of the project in the **Project Name** field.

(3) Verify that the default directory displayed in the **Location** field is the correct location, else click the **Browse** button to select another location.

> **Note:** The default project directory is configured in the General Preferences (Section 3.5.1) option.

(4) Check the **Add project to SVN** checkbox.

(5) Verify that the default URL displayed in the **Repository URL** field is the correct location, else click the **Browse** button to select another location.

(6) Click **OK.** (0)

The project Main Window is displayed with the new project tree displayed in the Project Tree pane.

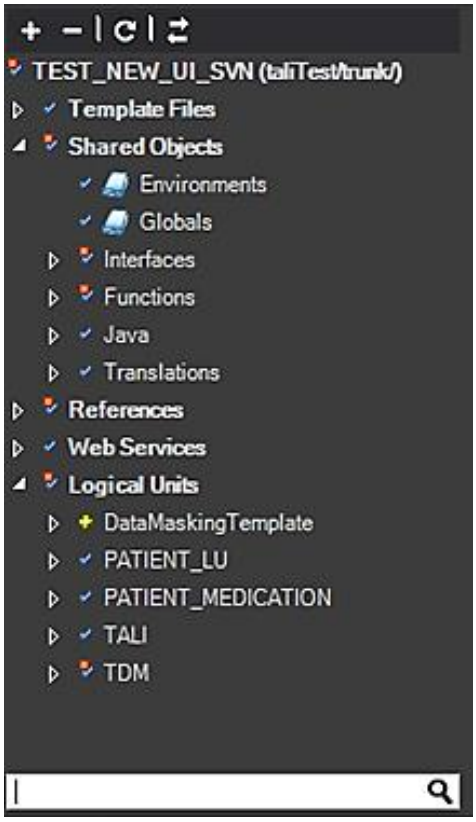
> **Note:** The project name is automatically added to the Repository URL.
>
> **Recommendation:** It is recommended that the first action after opening a new project is to save and commit the project baseline.

4.3.1 Project Tree Version Control Indicators

When a project is registered in the Configuration Control system, the following set of indicators appears in the Project Tree pane to identify the status of the project entities:

	Modified
	New objects
	Locked
	Updated
	Branch with modification
	File is versioned, but was inserted from a different location
	File is versioned and locally locked, but not modified
	File is versioned, locally locked, and modified
	File is in conflict; must be resolved before continuing

The following is an example of the SVN indicators in a project tree.



4.4 SVN Checkout

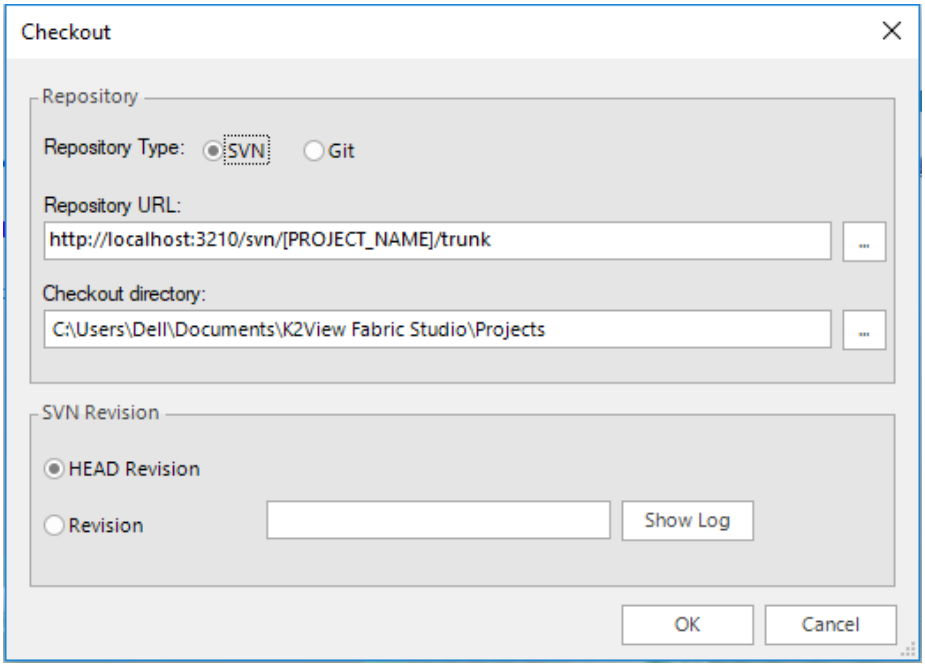
A standard feature in a SVN project is to enable multiple users to work concurrently on the same project and be aware of all changes through a locking and tracking mechanism.

Each user has a local copy of the project, which is synchronized with the central repository of the project. The synchronization is performed manually (see Section 4.5).

☞ To create a local copy of a Project

(1) Select **Checkout Project** from the File tab.

Checkout window is displayed.



(2) Edit the **[PROJECT_NAME]** text in the Repository URL field to be the name of the project to checkout.

(3) Verify that the directory in the **Checkout directory** field is correct, or browse to select the correct directory.

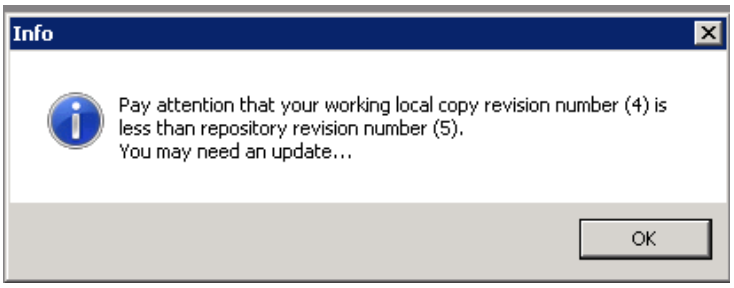
(4) Select the revision to check out in the **Revision** panel.

(5) Click **OK**. (0)

> **Note:** To avoid overrides and confusion, it is recommended that this location be a secure area.
>
> **Note:** The **Checkout Directory** default name is the project name as set in the Checkout directory. For example: `C:\Users\User1\Documents\K2Project\Company_Project`

> **Note:** If a previous revision is selected, click **Show Log** to view the log and select which revision to use.

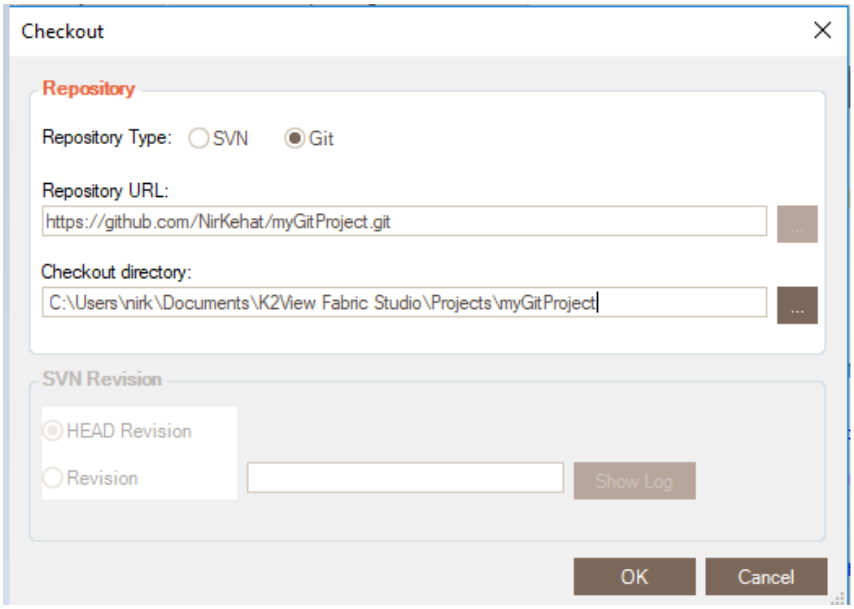
Whenever you open a project that has been updated and committed to the repository by another user, a warning message is displayed requesting confirmation that the revision on the local machine is up-to-date.



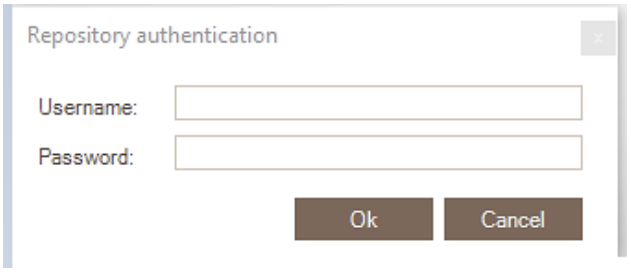
4.5 GIT Checkout

(1) Select Checkout project from Fabric studio main page.

(2) Add your GitHub repository URL: (In Checkout directory please remove Git extension).



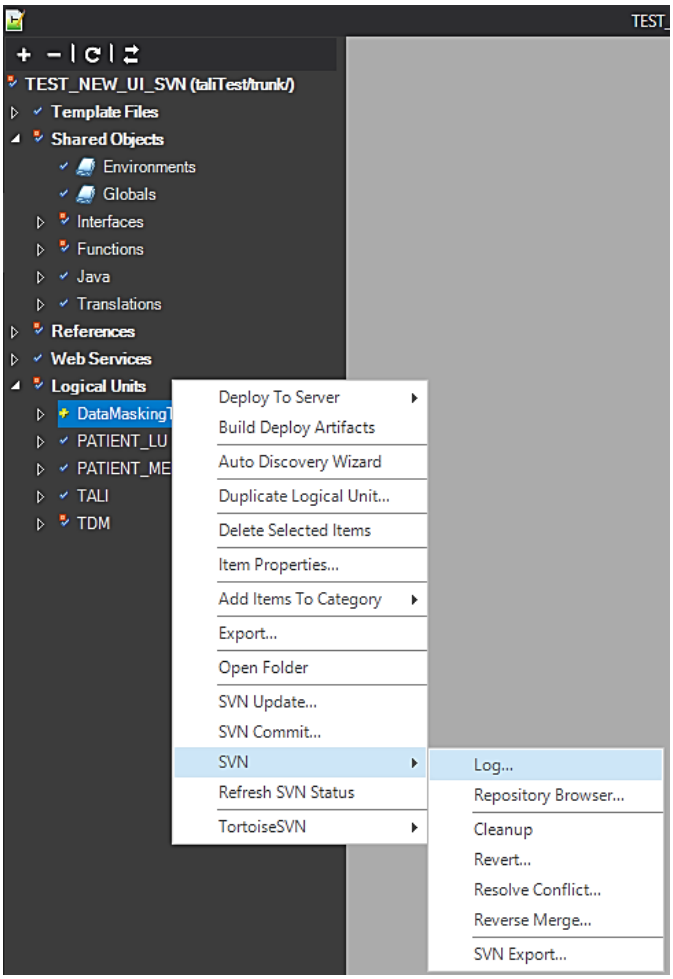
(3) Enter Username and Password.



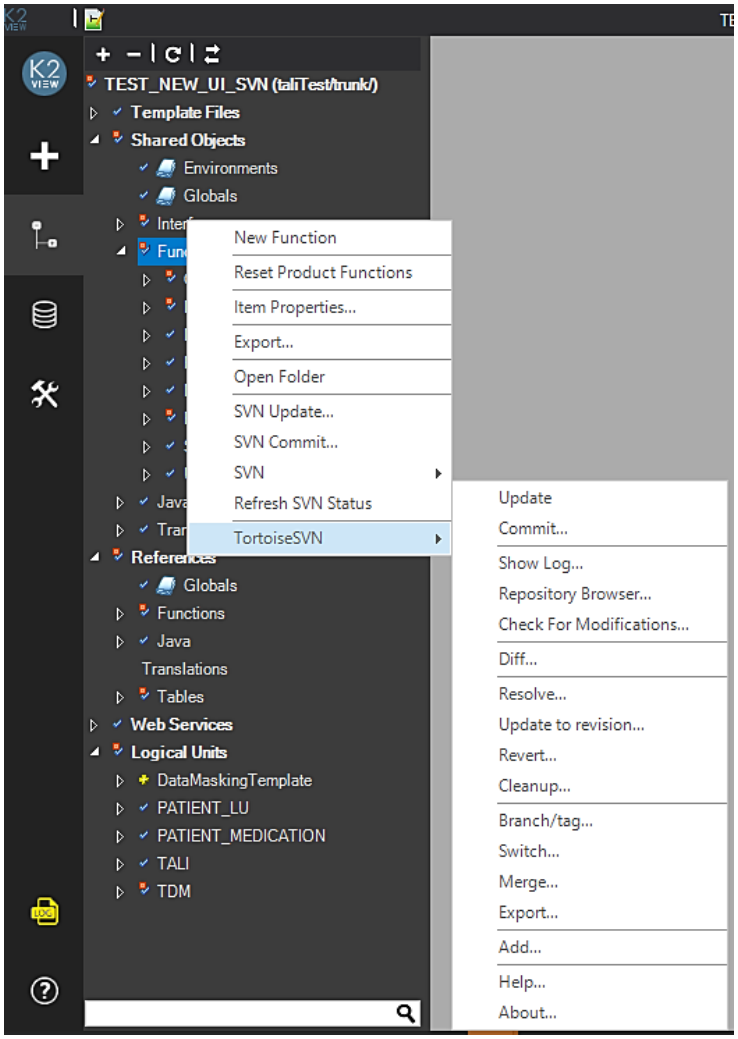
4.6 SVN Operations

The Project Tree context menu for project objects includes options to perform SVN operations. These operations are relevant to a project that is registered with the Configuration Control repository.

The context menu displays two top-level operations:



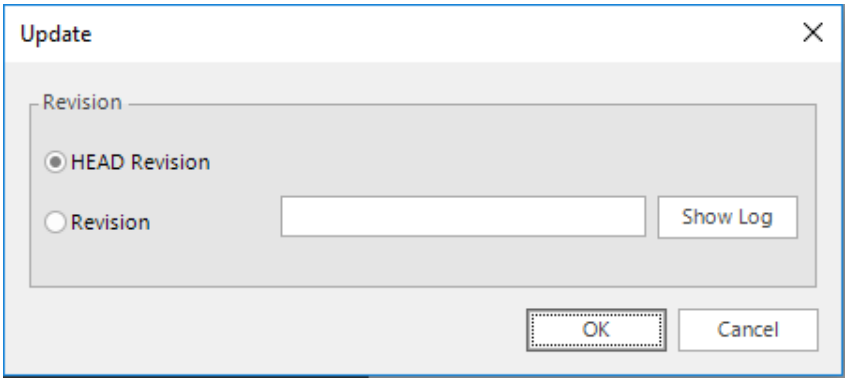
In addition, the Support TortoiseSVN commands directly from the project tree.



- **SVN Update**: Since a typical user works on a local machine, the best practice is to synchronize the local copy with the central repository. The Update option enables users to perform manual synchronization at any time.

  You can choose to update to the HEAD Revision (latest) or to an earlier revision.

  K2View recommends updating to the HEAD Revision, unless there is a specific reason (for example, an urgent bug-fix) to update to a specific earlier revision.



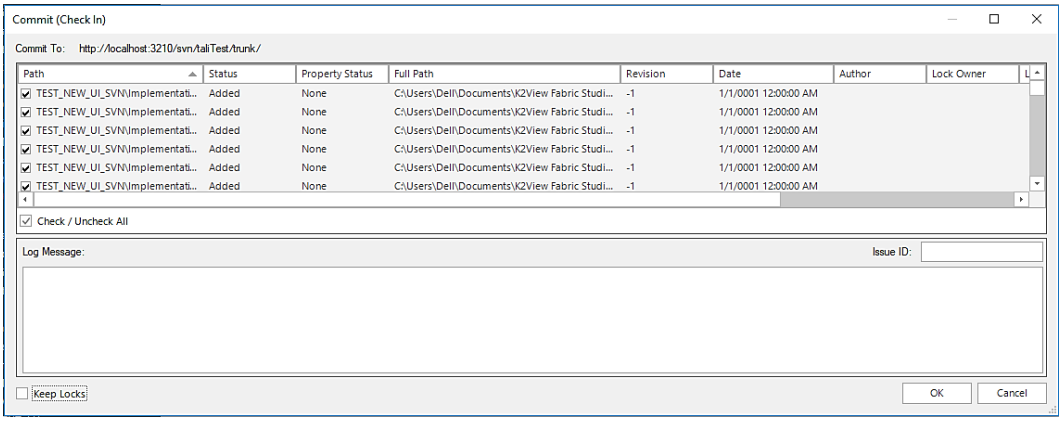
After the update, a message is displayed confirming that the local copy of the project has been updated.

- **SVN Commit:** When the implementation of a component, or set of components, is complete, use this option to commit all changes to the central repository. The committed components are available to other users. You can commit individual components or all components.

  K2View recommends that a log message (including a reference to any Issue ID that may be relevant) be generated to document the changes that are being committed.

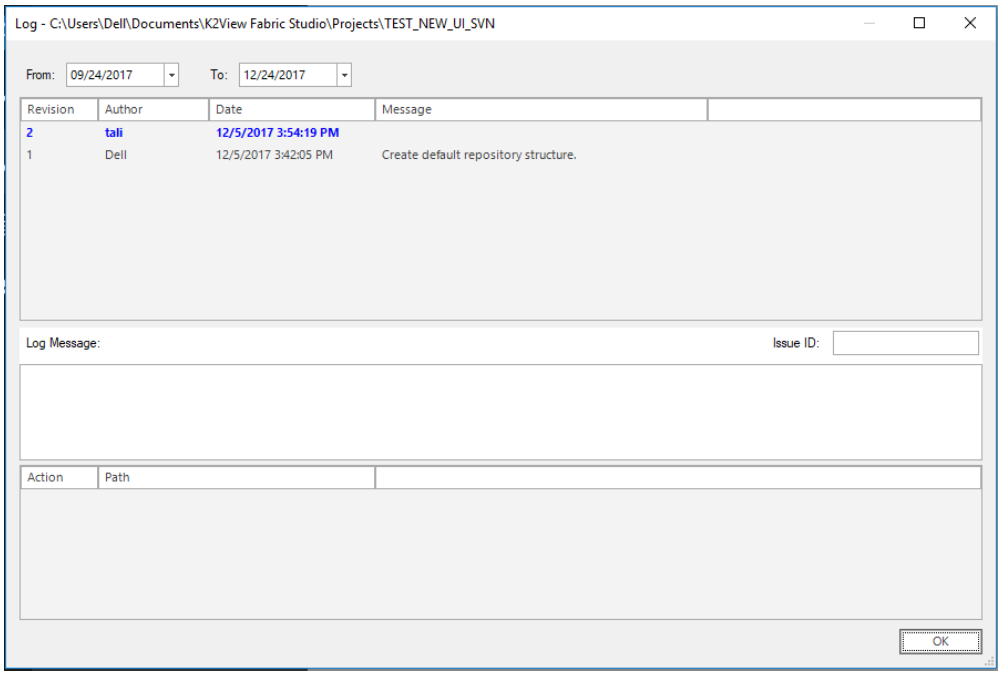
  Even after a component is committed, you can still lock it (see below) to prevent others from editing it.

  > **Note:** Locking a component may cause conflicts later and should only be used in situations when further changes must be made.



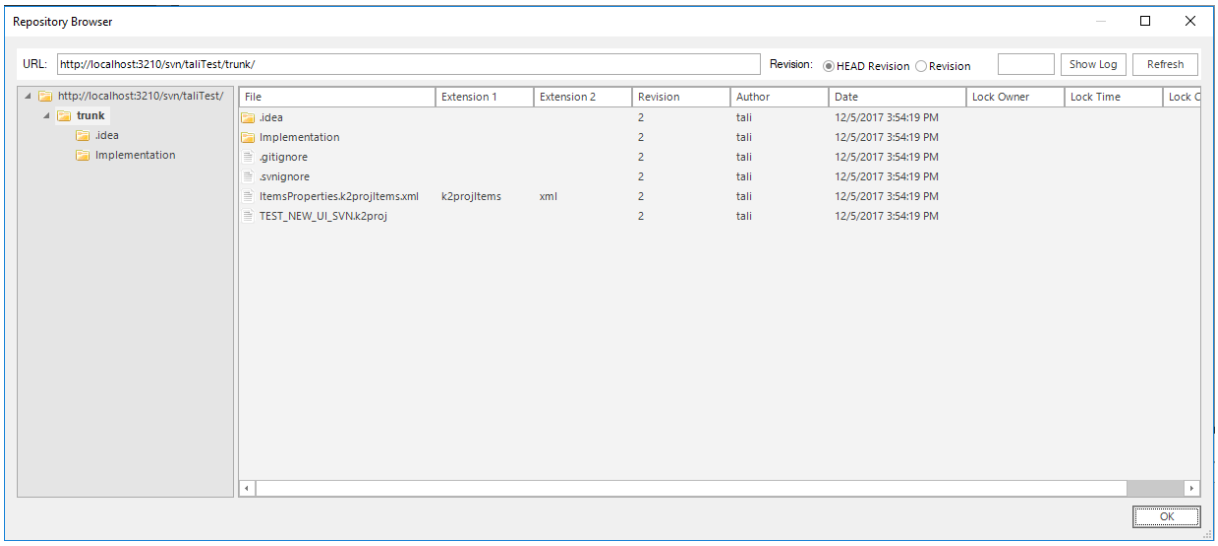
The second-level SVN options (available when selecting **SVN** in the context menu) are:

- **Log**: Allows you to review the changes for a specific component, level in the tree, or over a given period. Displays a Log window that lists the log entries associated with the object.

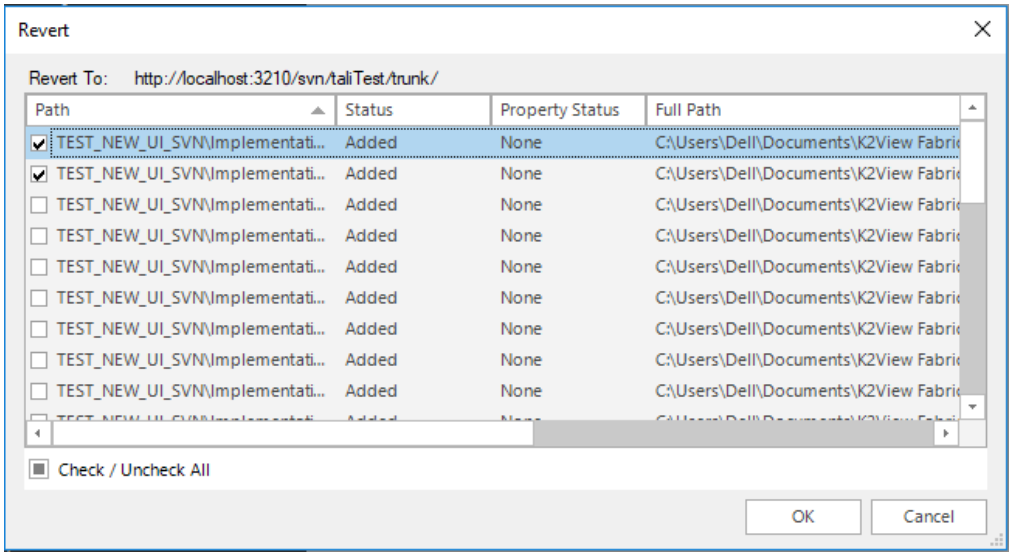


You can sort the columns by clicking the column headers.

- **Repository Browser**: Displays a window that enables you to browse different versions of the project or a specific object. By default, the Head (latest) revision is displayed, but it is possible to drill down to the level of a component and review the log.

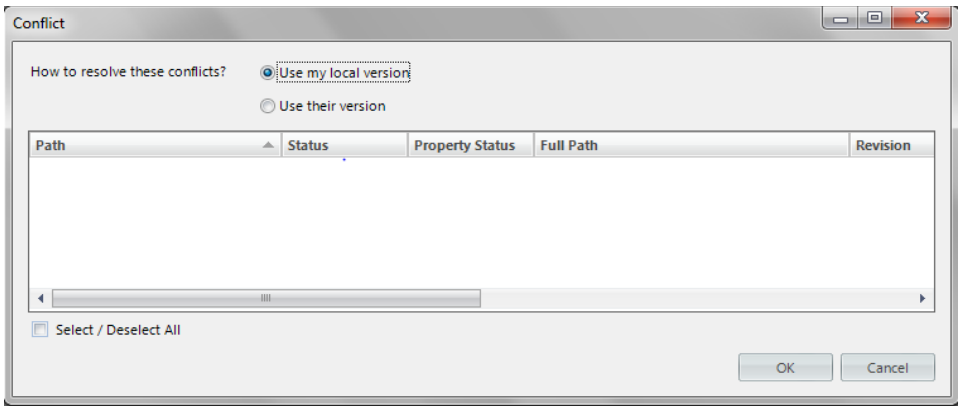


- **Cleanup:** enables you to perform a sanity check on the SVN issues of the client. Upon completion, a message box is displayed to report success or identify problems.
- **Revert: E**nables you to roll back any changes that have not yet been committed and revert to the version from the SVN repository. By default, all changes under the tree component are rolled-back, however, it is possible to choose only selected changes.



- **Resolve Conflict:** Enables you to resolve a conflict between the local version and the version in the SVN. These conflicts may arise if a new project version committed by another user contains changes conflicting with the locally made changes.

  You can select to resolve the conflict based on the local version or the committed version.



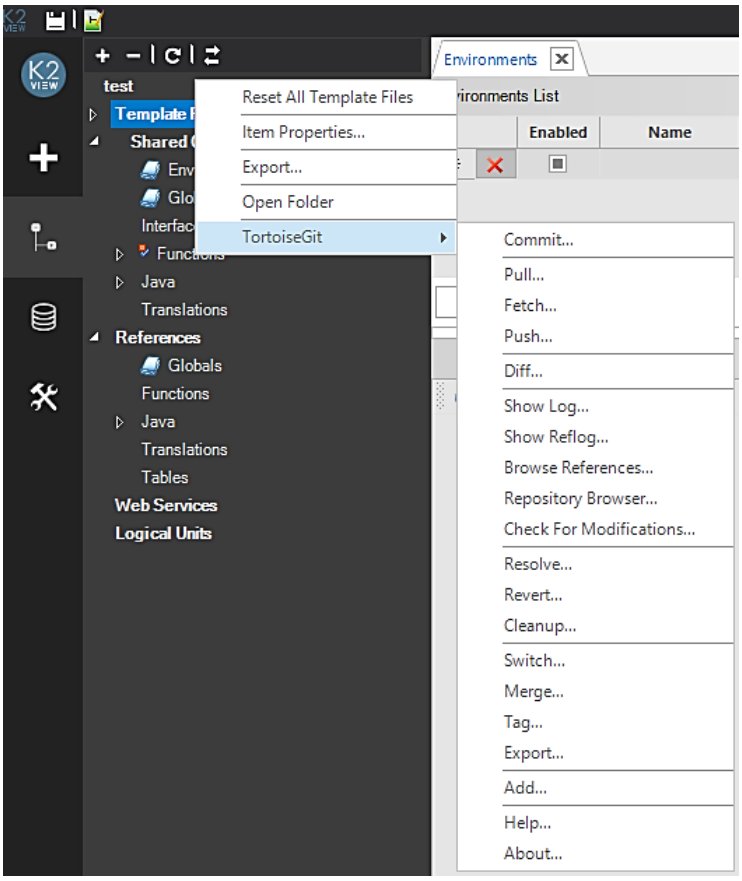
> **Note:** To downgrade the SVN version to a previous one, use the 'Update' operation to update the local version with the required previous version; Then use '**Reverse Merge**' in order to commit this version.

> **Note:** None of the changes committed in later versions are saved.

- **SVN Export:** Exports the SVN version of the entire project**.**
- **SVN Branch feature:** One of the features of version control systems is the ability to isolate changes onto a separate line of development. This line is known as a branch. Branches are often used to try out new features without disturbing the main line of development with compiler errors and bugs. As soon as the new feature is stable enough, the development branch is merged back into the main branch (trunk), accessed from the root project tree SVN context menu.

4.7 GIT Operations

The following operations are available using TortoiseGit:



4.8 Exporting and Importing

K2View Fabric Studio supports a proprietary XML-based format (k2export) for encapsulating the information regarding a project. This format encapsulates the entire project structure and components. Using this format, you can save a project and later use its components to merge into a central project or move to a different workstation.

When exporting a complete project controlled by and committed to the SVN repository, you can select to export to a Tar.gz file.

4.8.1 Exporting Projects or Specific Objects

You can export the entire project, a branch within a project, or a specific object. This functionality can be utilized, for example, to create a branch of a generic product, for customized projects, or customized layers.

(1) To export a project, right click the project root or top branch, and select Export… from the context menu.

(2) After selecting the **Export** option, you are prompted to select the exported file location.

4.8.2 Importing Project or Specific Objects

You can import either a complete exported project or specific branches of the project tree. The imported objects are attached to an existing project.

> **Note:** Importing a project exported from a higher version of K2View Fabric Studio is not supported.

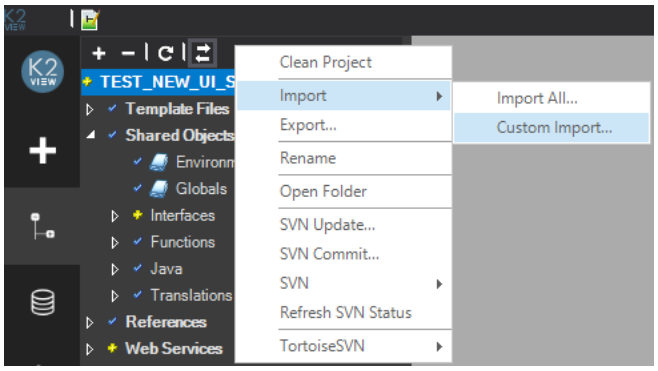
☞ To import a specific branch of a project

(1) Right-click the **Project Tree** root object.

Context menu is displayed.

(2) Select the **Import** option.

Second level menu is displayed.



(3) Select the **Custom Import** option.

File browser window is displayed to select the k2export file to be imported.

(4) Select the file and click **Open**.

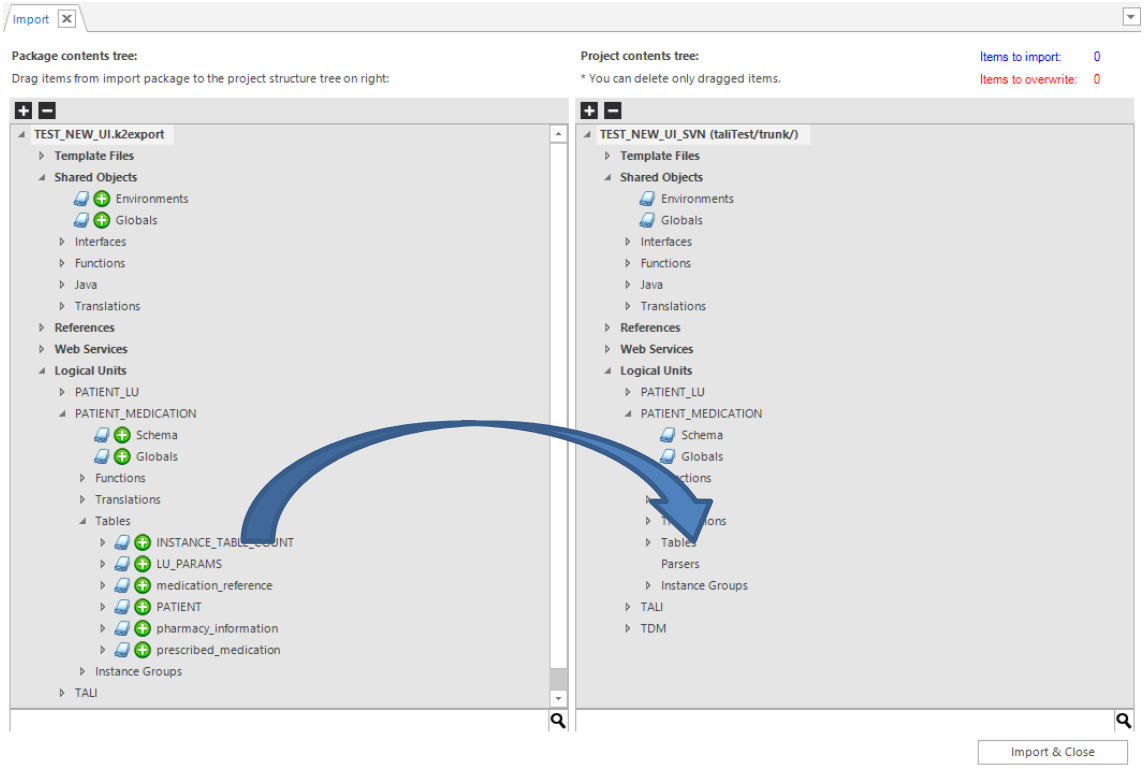
Custom import window is displayed.

(5) Select the branches and items to import from the file.

The selected items are indicated by the  *icon.*

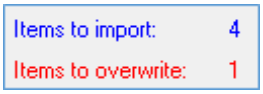
(6) Click and drag the  to the current project structure, displayed in the right side pane.

The cursor changes to the  *icon to indicate a legal landing spot for the imported entities.*



(7) Release the mouse button.

The import counters reflect the number of entities to import and overwrite.



(8) Click Import & Close when completed indicating what entities to import.

K2View Fabric begins transfer of objects to a new location.

> **Note:** The selected Objects can only be imported to the right section of the project, either on the project or Logical unit level.
>
> **Note:** If importing objects with the same name and location in the project tree, a notification icon appears next to the imported objects ⚠. Importing those objects overrides the existing ones.

Exceptions:

(1) In order to Import a function from a category that does not exist in the new project, drag the function to the '**Functions**' item itself – this action creates a new category in the target project along with the required function.

Importing Globals to an existing Globals file does not override the existing ones but merges both files into a single file.

(2) Right-click the Target Project tree to display a context menu that allows actions on the imported items:

- **Remove All Conflicted Items** –removes all the conflicted items from the target, leaving only new items.
- **Remove Import Items** –removes only the selected items (right click to select an item).
- **Remove All Items** - removes all items from the target.
- **Show conflicted items only** –displays only the items that conflict with the existing items.
- **Show Import item only** –displays only the items being imported.
- **Show all items** – displays all items in the project tree.

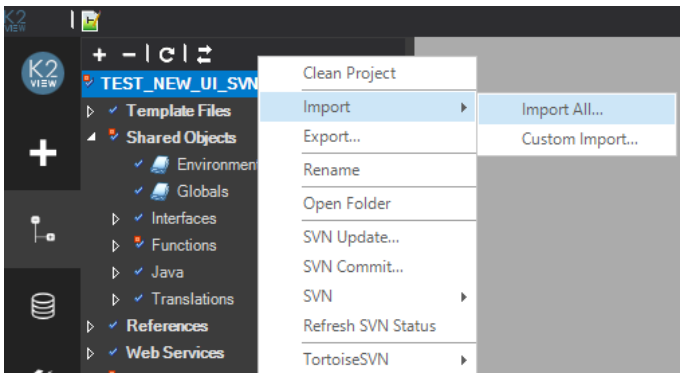
☞ To import the entire set of project entities

(1) Right-click the Project Tree root object.

The Context menu is displayed.

(2) Select the Import option.

The Second level menu is displayed.



(3) Select the **Import All** option.

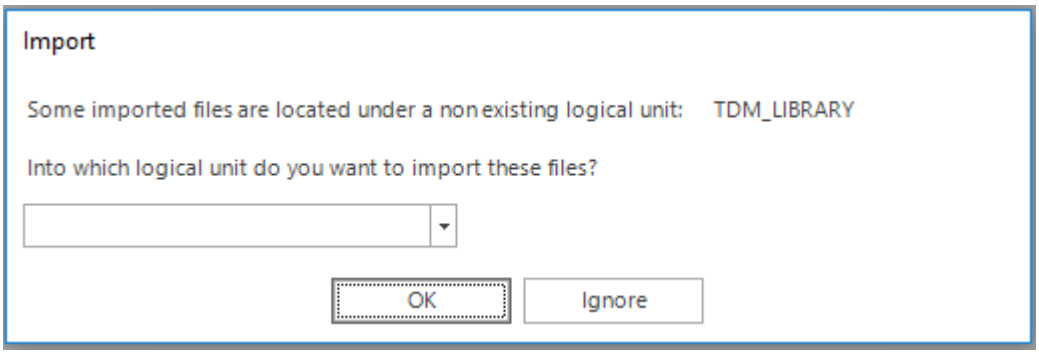
The File browser window is displayed to select the k2export file to be imported.

(4) Select the file and click **Open**.

The project import begins.

> **Note:** If the file name already exists, right click and there is a prompt to confirm that the user wants to override the file.

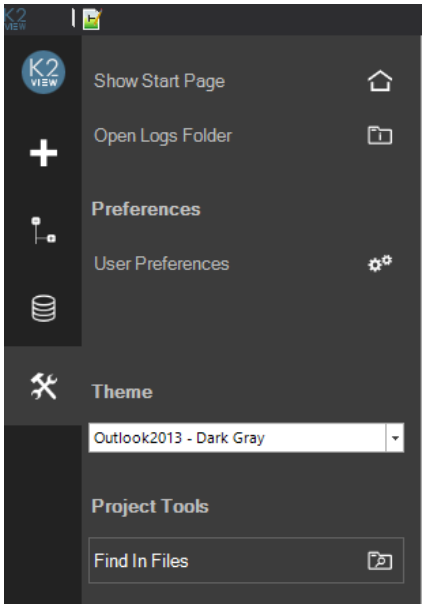
Note: When importing files to a non-existing LU, the user must choose the LU into which to import these files or to create a new LU named by the exported LU.



4.9 Searching and Replacing

You can search for string references in any object, function, expression, or target table. The search option can be accessed anywhere in the application by clicking <Ctrl+Shift+F>. In addition, you can replace all occurrences of a string with another string.

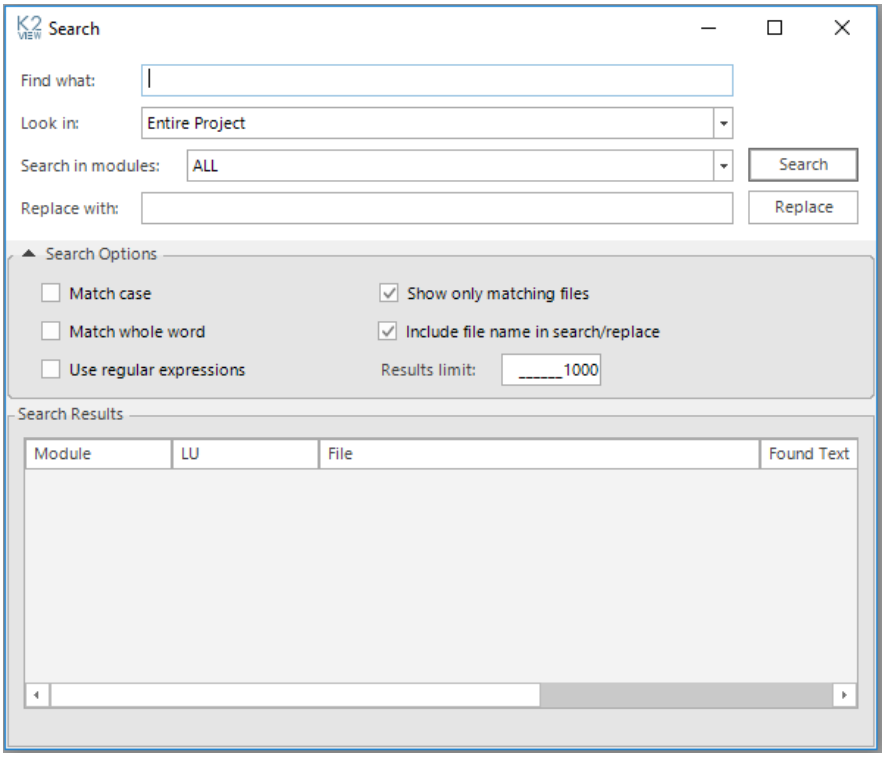
Another way to open the search window is by clicking the "Find In Files" option under the Tools tab.



To search for a String

(1) Press <Ctrl+Shift+F> or click the "Find in Files" option.

The Search window is displayed.



(2) Enter the string to find in the **Find What** field.

(3) Select the Project or Logical Unit from the **Look in** drop-down list to indicate the scope of the search.

(4) Select the modules to search in the **Search in modules** drop-down list.

(5) Select the relevant options in the **Search Options** area.

(6) Click Search.

*A list of locations where the String was found is displayed in the **Search Results** area.*

(7) Double-click the item to open the corresponding object.

To search and replace a String

(1) Perform a search, as described in steps 1–5 above.

(2) In the **Replace with** field, enter the string to be used instead of the current string.

(3) Click **Replace**.

A message confirming the replacement of all string occurrences appears.

(4) Click **OK** to confirm.

All modules are closed and the string occurrences are replaced.

⚠ ***Caution:*** *The Replace option might damage your project. Please use this option cautiously.*

4.10 Log Files and Error Tracking

K2View Fabric Studio incorporates a testing and logging capability. This enables users to develop, test, and monitor the test log from within the application.

The Server/Activity Logs icon is located in the left menu of the Fabric window:

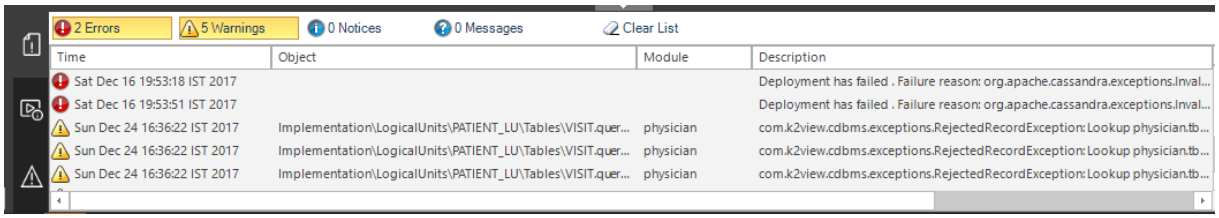
4.10.1 Server Logs

(1) Click the Server/Activity Logs icon to display a scrolled window, listing all the recent messages from the server side. By default, this window only displays warnings and error messages.

(2) Click on the server logs icon: 

(3) Click a Notice or Message to view the entire notice or message.

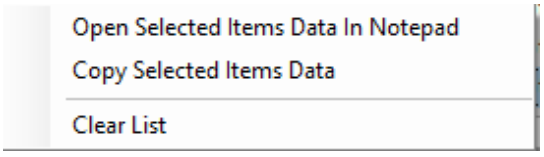
(4) Clicking any of the message type toggle buttons (along the top of the pane) filters the corresponding messages types.



(5) Double-click the message in order to open the component where the error was generated.

(6) Click the Clear List button Clear List to clear the text board of previous messages.

(7) Right click in the messages area to display the following context menu:



(8) Select **Open Selected Items Data in Notepad** and review the full error or notice message.

(9) Select **Copy Selected Items Data** in order to copy the full text into the clipboard for further tasks.

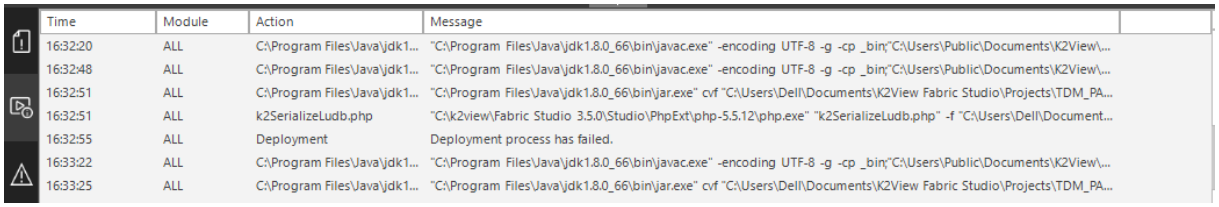
(10)Select **Clear List** in order to clear the text board of previous messages.

4.10.2 Activity Logs

(1) Click the Server/Activity Logs icon to display a scrolled window, listing all the recent messages from the server side. By default, this window only displays warnings and error messages.

(2) Click the Activity Logs icon: 

The following window is opened:

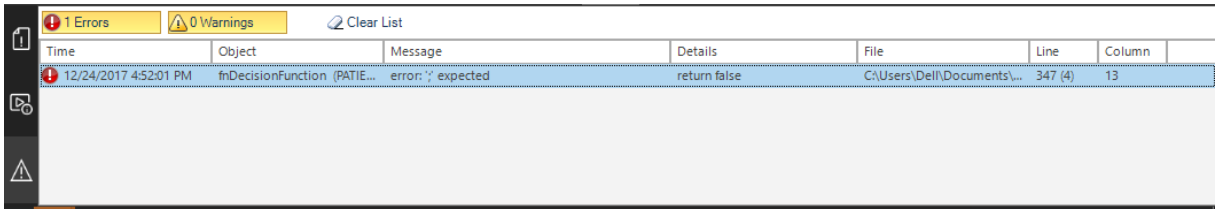


4.10.3 Compilation Errors

(1) Click the Server/Activity Logs icon to display a scrolled window, listing all the recent messages from the server side. By default, this window only displays warnings and error messages.

(2) Click the Compilation Errors icon: 

(3) The following window is opened:



5 Working with Database types

5.1 Overview

Database type was added as a part of Fabric 5.1 in order to add the ability of supporting additional database types without product enhancements.

A user can define a new database type according to the database and JDBC driver specifications.

When introducing a new Database Type, the user should locate the JDBC driver files under the project directory/lib/<new database type>/.

It is also possible to override one of the database types provided as a part of the product package, if some customizations are required.

5.2 Adding a new Database Type

☞ **To create a new Database Type:**

(1) Right-click the **Database Types** in the project tree, and then select the **New Database Type** option in the context menu.

(2) Assign a Name to the new Database Type.

(3) Assign Class Name.

(4) Select URL Template.

(5) Select the default port (not mandatory).

(6) Set Pool Properties – should be modified only by advanced users. As initial setup, use the default values.

(7) Set Fabric properties - should be modified only by advanced users. As initial setup, use the default values.

> **Note: to connect REAL field (Oracle) to a TEXT field (PostgreSQL) in your LU- you need to set the 'Support Implicit Conversions' parameter of Fabric Properties to 'False' (the default value it 'True').**

(8) Studio – Metadata – in case JDBC driver doesn't provide access to meta data, fill in the following:

- SQL Query for Query for Tables List
- SQL Query for Column List
- SQL Query for FKs List- define custom SQL query to retrieve the list of foreign keys (FK) and primary keys (PK) of the tables. These keys can be

used by the auto-discovery wizard when creating a new logical unit. The query must contain the following aliases:

- PKTABLE_SCHEM- set for the PK schema name
- PKTABLE_NAME- set for table name
- PKCOLUMN_NAME- set for PK column name
- FKTABLE_SCHEM- set for FK schema name
- FKCOLUMN_NAME- set for FK column name

- Custom SQL Example- MySql DB-
  - FK query:

    SELECT table_schema as PKTABLE_SCHEM,

    table_name as PKTABLE_NAME,

    column_name as PKCOLUMN_NAME,

    referenced_table_schema as FKTABLE_SCHEM,

    referenced_table_name as FKTABLE_NAME,

    referenced_column_name as FKCOLUMN_NAME

    FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE

    WHERE referenced_table_name Is Not Null
  - PK query:

    SELECT k.table_schema as PKTABLE_SCHEM,

    k.table_name as PKTABLE_NAME,

    k.column_name as PKCOLUMN_NAME

    FROM information_schema.table_constraints t

    JOIN information_schema.key_column_usage k

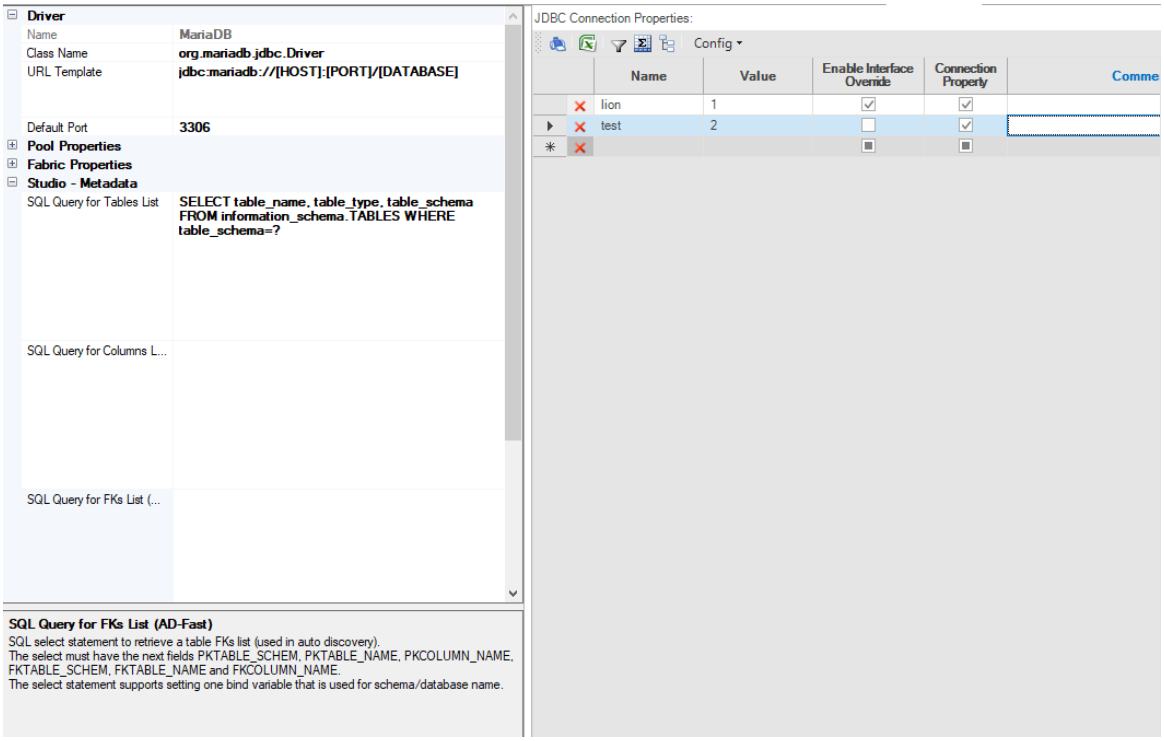
    USING(constraint_name,table_schema,table_name)

    WHERE t.constraint_type='PRIMARY KEY'

(9) Set Query Builder: identifiers Case Sensitivity - Insensitive or SenstiveUpperCase or SensitiveLowerCase

(10) Set JDBC connection properties – set Names and Values for parameters to be used as a part of JDBC connection properties when defining the interface based on the new Database Type. When setting the Enable Interface Override, user will be able to change the parameter value on the interface and it will appear under the Connection Properties section. When setting the Connection property, connection properties map will be sent to JDBC driver.

(11)Click the **Save** icon ( ) or press CTRL+S to save your changes.



> **Note:** you can include dots marks in the 'Name' column of the JDBC properties. For example - add the following connection for Oracle: oracle.jdbc.ReadTimeout=10000

5.3 Adding a new Database Type from Template

Creation of a new Database type from Template is required, when a user wants to customize a product build-in database type.

To create a new Data Source Interface:

(1) Right-click the **Database Types** in the project tree, and then select the **New Database Type From Template** option in the context menu.

(2) Edit the required properties.

(3) Click the **Save icon ( )** or press CTRL+S to save the changes.

5.4 Editing Database Types Settings

☞ **To change an existing Database Type:**

(1) Double-click the Database Type in the Project Tree pane.

(2) Edit the required properties.

(3) Click the **Save icon (**  **)** or press CTRL+S to save the changes.

5.5 Deleting Database Type

☞ **To delete a defined Database Type**

(1) Right click the name of the Database Type (under Shared Objects/Database Types) in the Project Tree.

(2) Select **Delete Selected Items** from the menu.

6 Working with the Data Source Interface

6.1 Overview

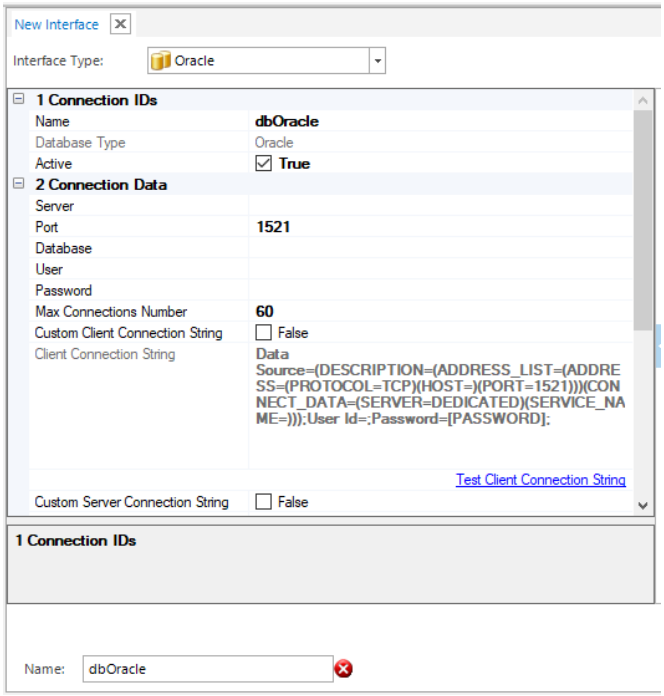
The Data Source Interface defines the connections to the legacy databases that are the sources for data within K2View Fabric. The Interface includes:

- An identification of the database server, meaning its IP address, and a communication port for importing data.
- A database type, for example Oracle, SQLite, DB2, Mysql, Fabric. This determines the structure of the queries generated by the Studio.
- The database name
- The authorization credentials for making the connection

The Interface is used to access the existing database structure and the data that needs to be transmitted to the K2View Fabric server in an ongoing basis.

6.2 Interface Connection Parameters

A central component of configuring a data Interface is defining the database connection parameters for the source database. The exact list of connection parameters is dependent upon the database type.



The following table presents the most common required connection parameters shared by all databases:

Parameter	Description
Database Type	The database type, for example Oracle, DB2. Determines many of the default values for other parameters
Name	The file name of the database
Alias	Alias connection to the database
Server	Hostname or IP address of the database server
Port	Connection port for communication with the database. A default port is entered based on the Database Type chosen.
Database	Database, schema, or service name
Schema Filter	Database schema to access. Required if more than one schema exists.
User	Username used for database access authentication
Password	Password used for database access authentication
Max connection Number	Sets the maximum numbers of connections per interface.
Active	Indicates whether to check at runtime if the database connection is active. This parameter allows users to disable a connection so that the server side ignores the specified connection.
Custom Connection String	A connection string that is being used by the client side. It is generated according to the database type. For example, for DB2, the string consists of server=;database=;UID=;PWD=;. When this checkbox is enabled, the string is editable.
Custom Server Connection String	A connection string that is being used by the server side. It is generated according to the database type. When the checkbox is enabled, the string is editable.

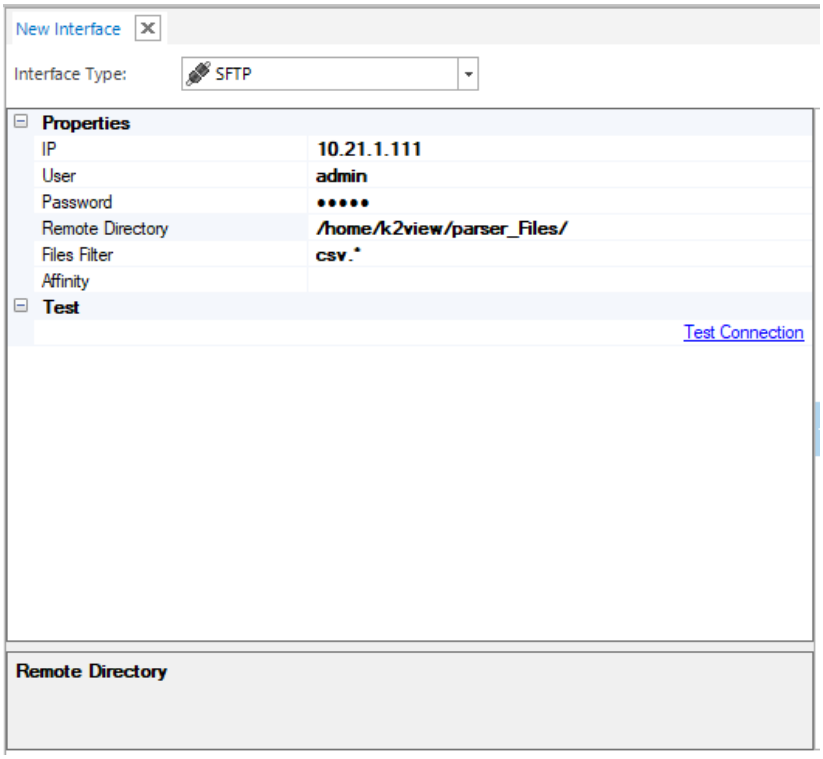
6.3 SFTP Interface

Ability to connect to ftp-server and stream a file content.

This ability is in a parser map.

To create a new SFTP interface:

- Click the **New DB** in the Activity Bar.
- Select the **SFTP Interface Type** from the drop-down menu.

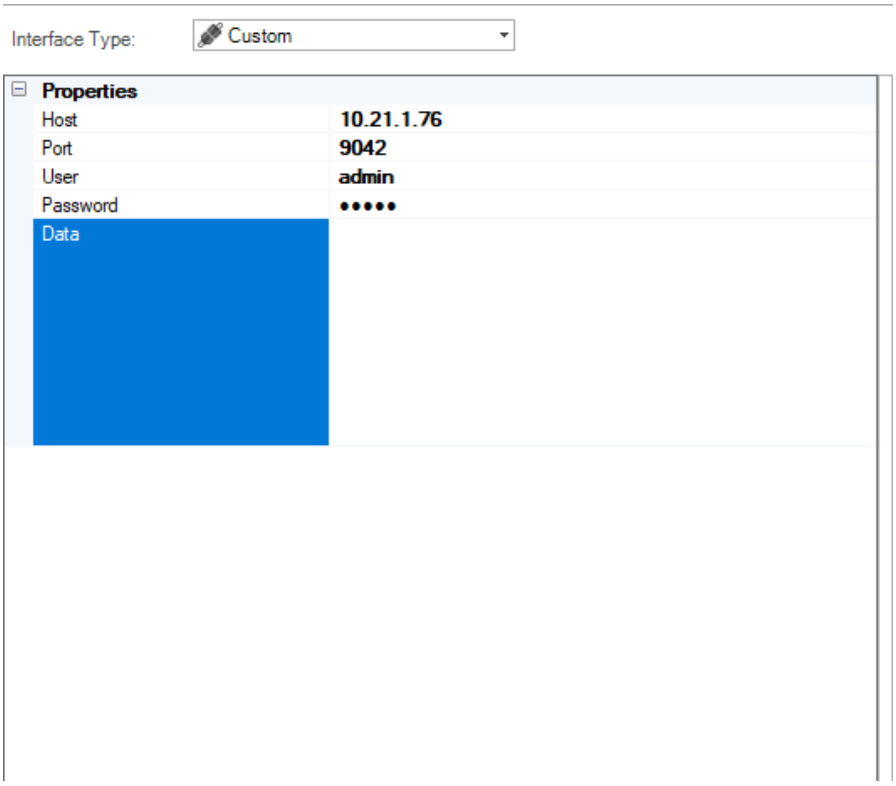


Parameter	Description
IP	Hostname or IP address of the FTP server
User	User name
Password	Password
Remote directory	The directory where files are stored
Files filter	Type of file

Parameter	Description
Affinity	An optional parameter. You can populate this parameter by IP address of a Fabric node, DC name, or by a logical identifier for Fabric nodes. Fabric allocates an execution server based on the populated affinity. If the affinity is not populated, Fabric allocates one of the available servers for the execution. Example for affinity: '10.21.1.169', 'DC1'

6.4 Custom Interface

Custom interface type to be used to interact with interface types that are not defined as a dedicated interface type in Fabric, such as SSH.



Parameter	Description
HOST	Hostname or IP address of the server
Port	PORT id
User	User Name

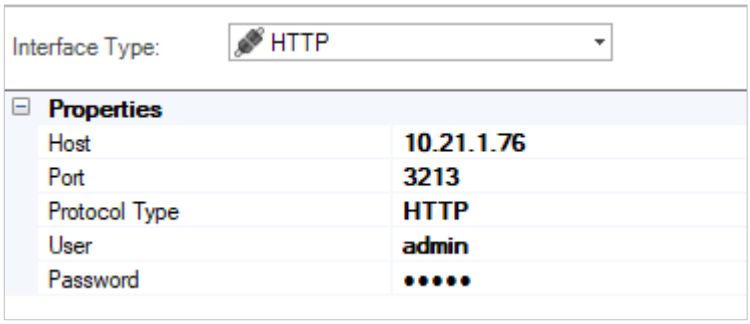
Parameter	Description
Password	Password
Data	Any additional parameters defined for the customer interface

6.5 HTTP Interface

Ability to connect to http interface. This ability will be in a parser map.

To create a new SFTP interface:

- Click the **New Interface** in the Activity Bar.
- Select **HTTP Interface Type** from the drop-down menu.



Parameter	Description
Host	Hostname or IP address of the HTTP server
Port	Port of the HTTP server
Protocal Type	HTTP or HTTPS
User	User name
password	Password

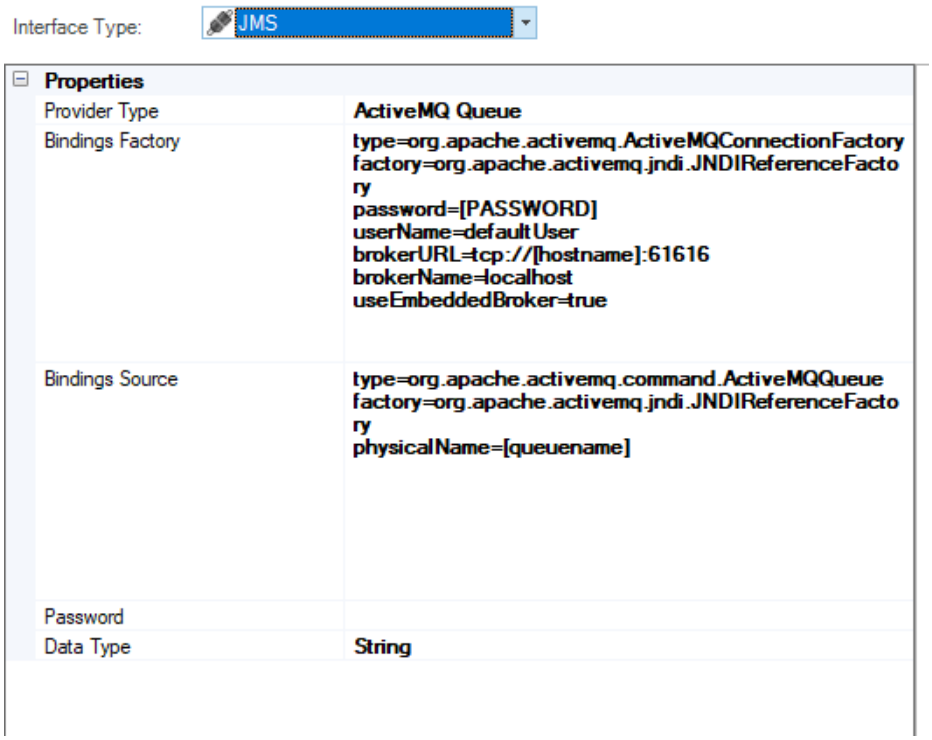
6.6 JMS Interface

Ability to connect to Java message queue. This ability will be in a parser map.

The JMS interface supports ActiveMQ, RabbitMQ or MQ queue or topic or custom provider queue/topic.

To create a new JMS interface:

- Click the **New Interface** in the Activity Bar.
- Select the **JMS Interface Type** from the drop-down menu.



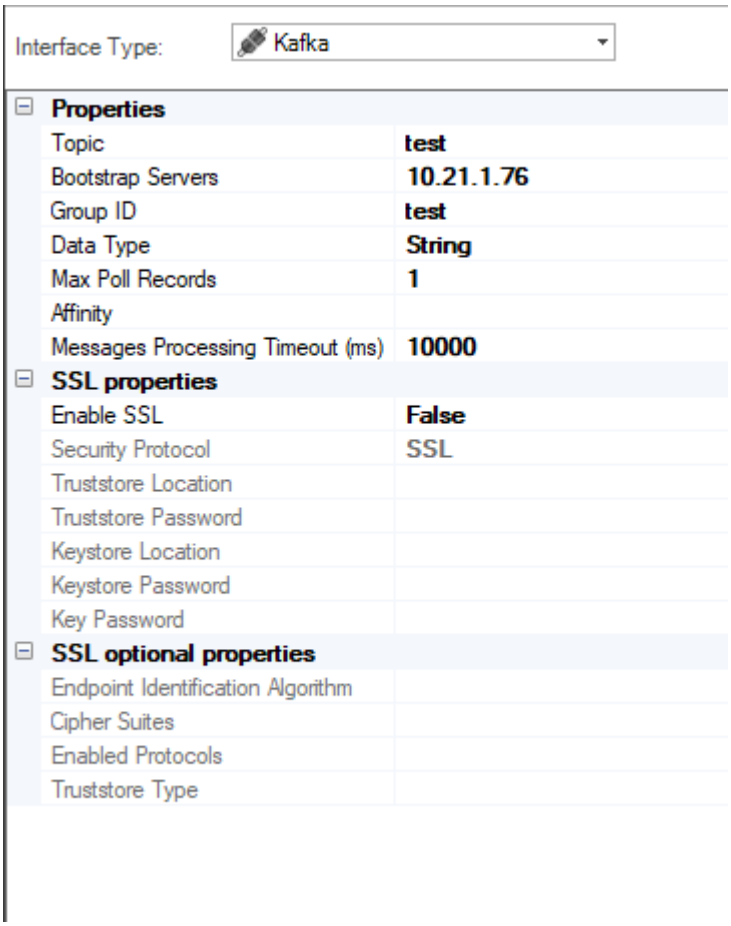
Parameter	Description
Provider Type	Supporting ActiveMQ, RabbitMq, MQ queue or topic Or Custom provider queue/topic
Bindings Factory	Bindings Factory
Bindings Source	Bindings Source
Password	Password
Data Type	String or Byte []

6.7 KAFKA Interface

Ability to connect to KAFKA queue.

This ability wilis in a parser map.

- Click the **New Interface** in the Activity Bar.
- Select the **KAFKA Interface Type** from the drop-down menu.



Parameter	Description
Topic	KAFKA topic name
Bootstrap Servers	Server IP
Group ID	KAFKA group id
Data Type	String or Byte[]
Max Poll records	Maximum poll records

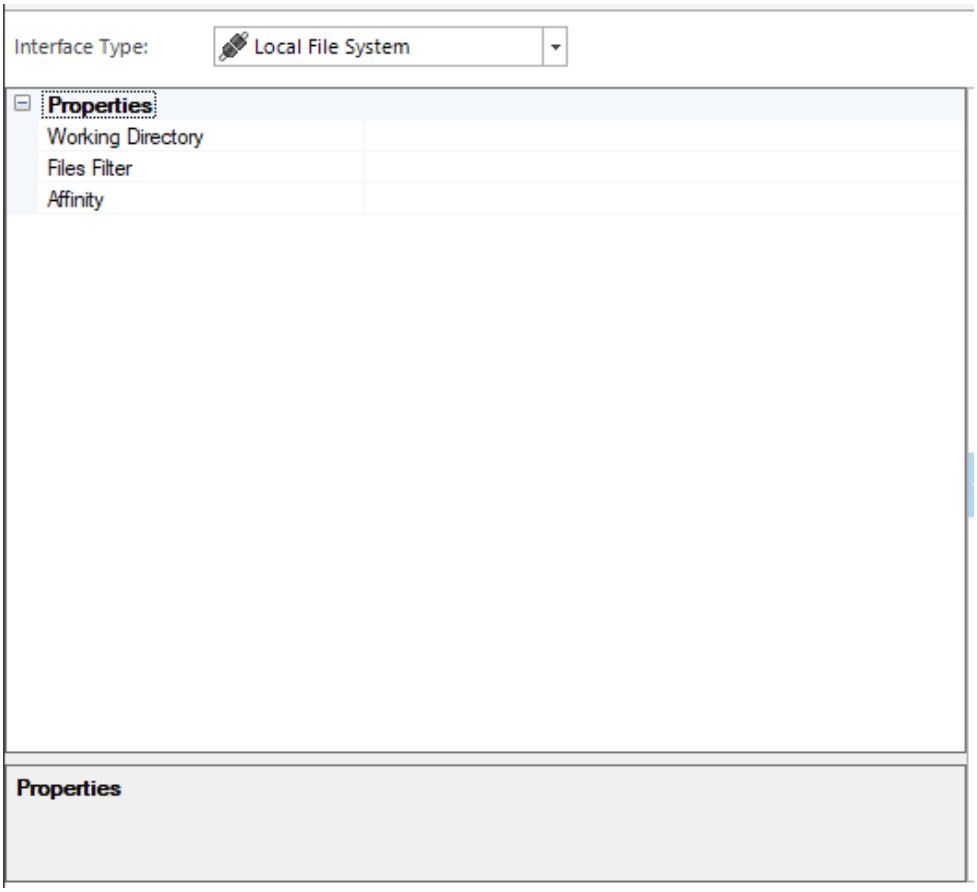
Parameter	Description
Affinity	An optional parameter. You can populate this parameter by IP address of by a Fabric node, DC name, or logical identifier for Fabric nodes. Fabric allocates an execution server based on the populated affinity. If the affinity is not populated, Fabric allocates one of the available servers for the execution. Example for affinity: '10.21.1.169', 'DC1'
Messages Processing Timeout(ms)	Timeout interval for messages processing
SSL properties	Properties that are mandatory if KAFKA is defined with SSL
SSL optional properties	Properties that are optional if KAFKA is deifned with SSL

6.8 Local File System Interface

This interface type supports processing files from a local directory of a specific Fabric server and can be used by parsers.

To create a new SFTP interface:

- Click the **New DB** in the Activity Bar.
- Select the **Local File System type** from the drop-down menu.



Parameter	Description
Working Directory	The directory where files are stored
Files filter	Type of file
Affinity	An optional parameter. You can populate this parameter by IP address of a Fabric node, DC name, or by a logical identifier for Fabric nodes. Fabric allocates an execution server based on the populated affinity. If the affinity is not populated, Fabric allocates one of the available servers for the execution. Example for affinity: '10.21.1.169', 'DC1'

6.9 Search Engine Interface

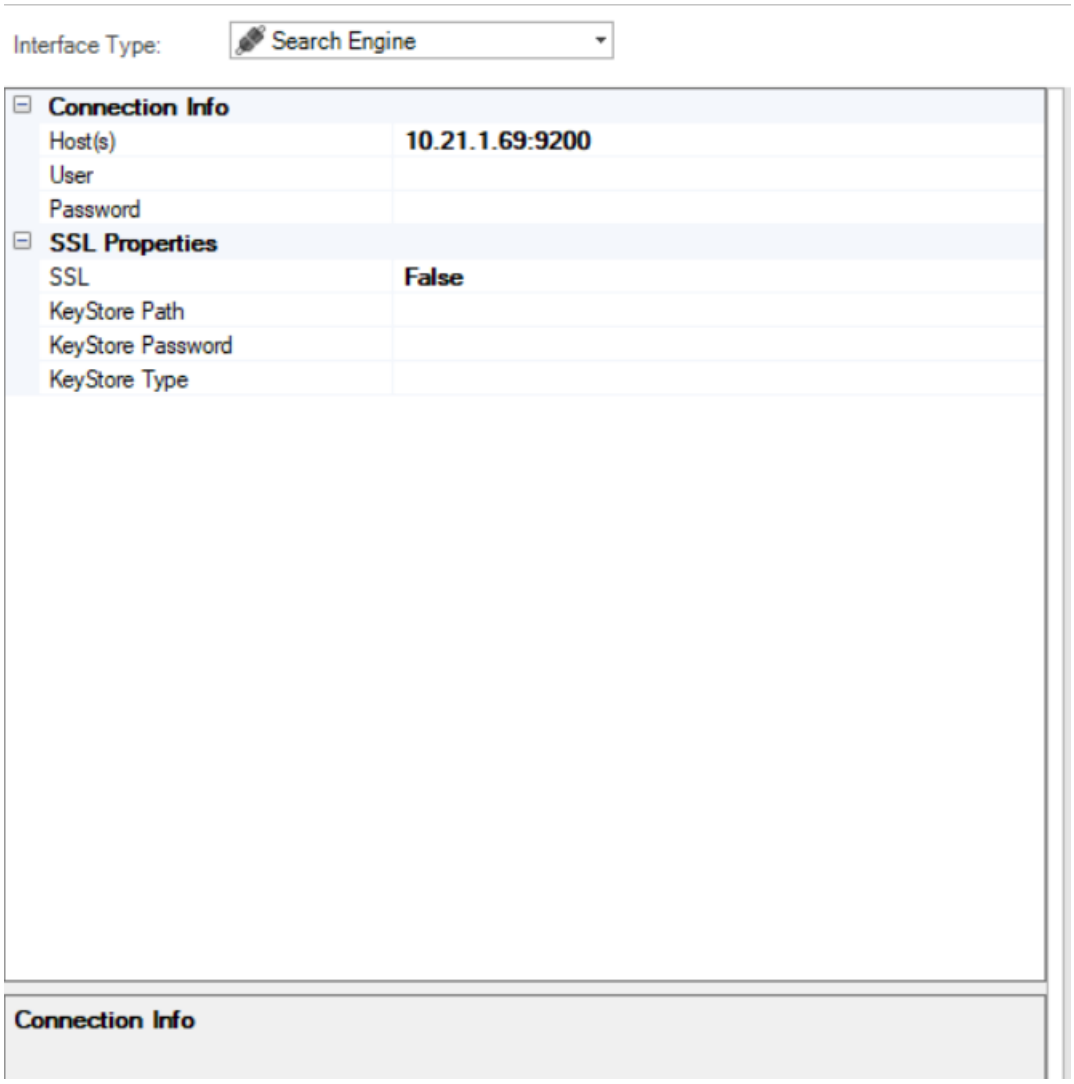
This interface type supports cross instance search functionality.

You must create a Search Engine interface and populate it with the connection details of the Elasticsearch for your search. This interface is needed to enable Fabric connecting the search engine when running the search command.

To create a new Search Engine interface:

(1) Click the **New Interface** in the Activity Bar.

(2) Select the **Search Engine** from the drop-down menu.



Parameter	Description
Host(s)	Comma delimited list of host and port of the ElasticSearch server

Parameter	Description
User	
Password	
SSL	By default- set to 'False'.
KeyStore Path	Key Store path in case SSL is set to 'True'
KeyStore Password	Key Store Password in case SSL is set to 'True'
KeyStore Type	Key Store Type in case SSL is set to 'True'

6.10 Adding a Data Source Interface

In Fabric 5.1, it is still supported only for existing interfaces.

☞ **To create a new Data Source Interface:**

(1) Right-click the **Interfaces** in the project tree, and then select the **New Interface** option in the context menu.

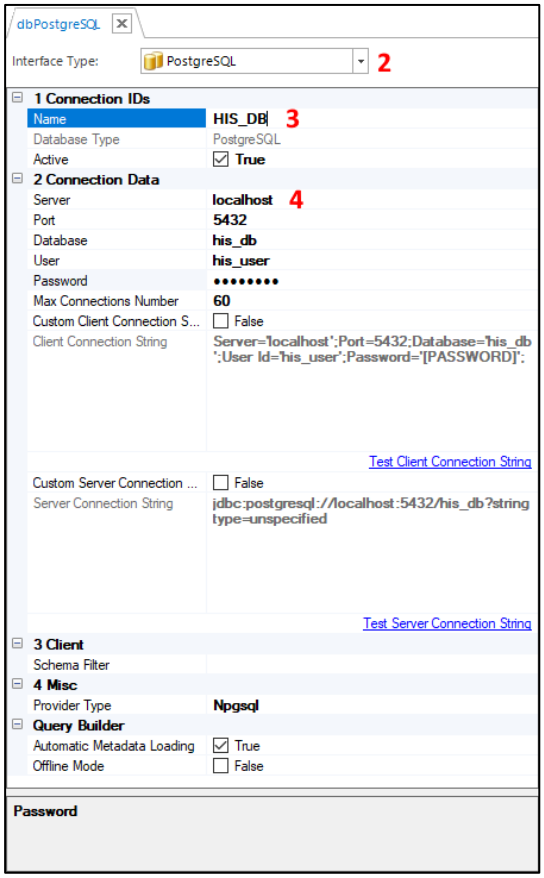
(2) Select the **Interface Type** from the drop-down menu (at the top of the window).

(3) Enter a name for the Interface file in the **File Name** field

(4) Enter the IP address or Hostname of the database server to connect to in the **Server** field.

(5) Optionally, enter values for the other Interface Connection Parameters (see Section 6.2, for details).

(6) Click the **Save** icon ( ) or use CTRL+S to save your changes.



Before closing the new interface definition, verify that the connection parameters are properly set, by testing the connection (see Section 6.13).

6.11 Adding a Data Source Interface – New Window – Fabric 5.1

To create a new Data Source Interface:

(1) Right-click the **Interfaces** in the project tree, and then select the **New Interface** option in the context menu.

(2) Select the **Interface Type** from the drop-down menu (at the top of the window).

(3) Enter a name for the Interface file in the **File Name** field

(4) Enter the IP address or Hostname of the database server to connect to in the **Server** field.

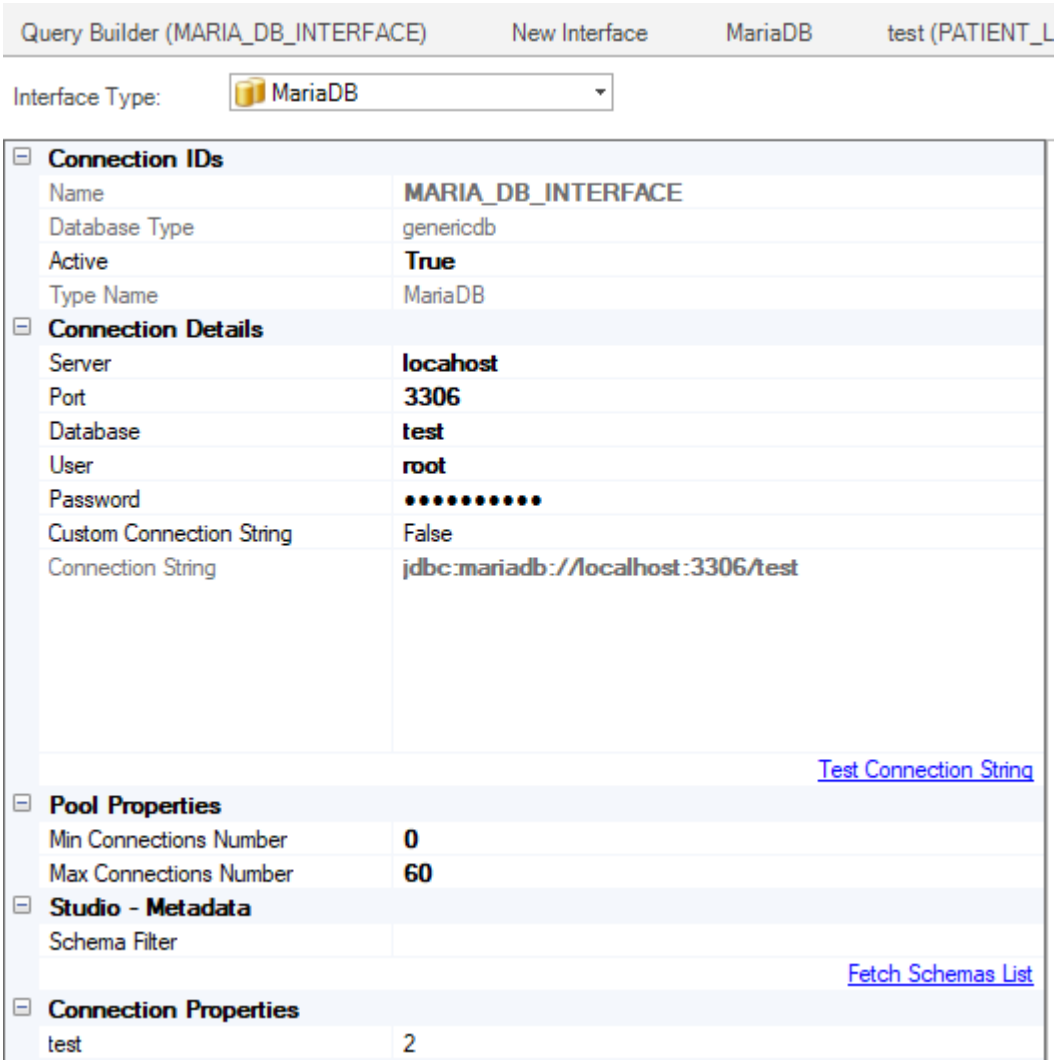
(5) Fill-in connection pool properties (or keep the default values)

(6) Optional – Set Studio – Metadata – Schema Filter – in order to set a filter on list of schemas to be used in Query-Builder and DBQueries.

(7) Option – Connection properties – in case JDBC connection properties were defined on the Database Type and it was marked as 'Enable interface override'.

(8) Only one Test connection will appear, as both Client and Server connection uses the same JDBC driver.

(9) Click the **Save** icon ( ) or use CTRL+S to save the changes.



Before closing the new interface definition, verify that the connection parameters are properly set, by testing the connection (see Section 6.13).

6.12 Adding Internal Fabric as a Data Source Interface

☞ **To create a new Data Source Interface:**

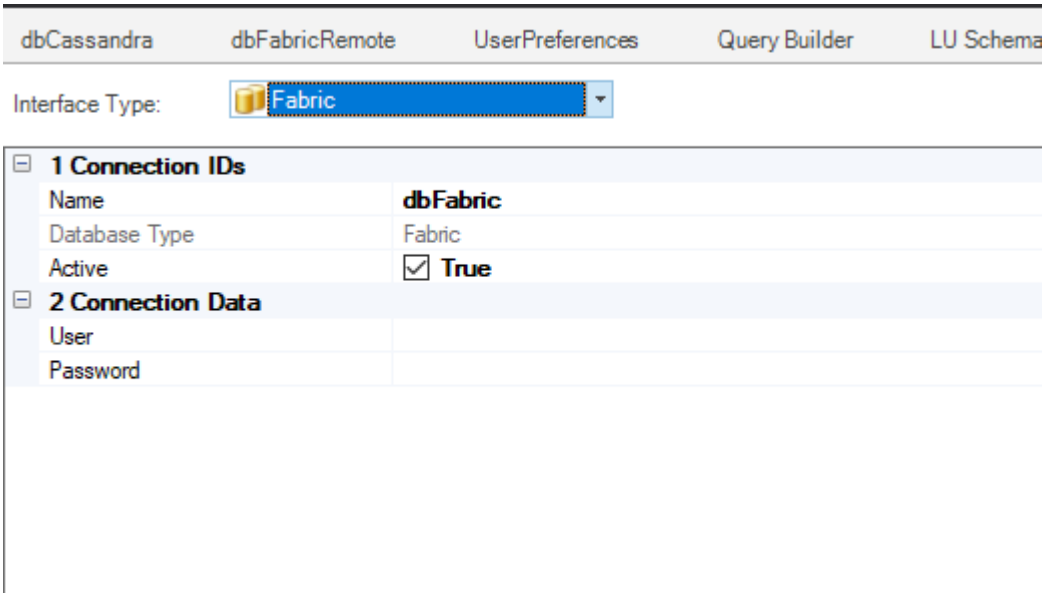
(1) Right-click the **Interfaces** in the project tree, and then select the **New Interface** option in the context menu.

(2) Select Fabric as **Interface Type** from the drop-down menu (at the top of the window).

(3) Set user and password for the internal Fabric interface.

(4) From release 5.0 and on it is a must to define Fabric as internal interface when using data from Fabric on Parsers, etc …

(5) Click the **Save** icon ( ) or press CTRL+S to save the changes.



6.13 Adding Remote Fabric as a Data Source Interface

☞ **To create a new Data Source Interface:**

(1) Right-click the **Interfaces** in the project tree, and then select the **New Interface** option in the context menu.

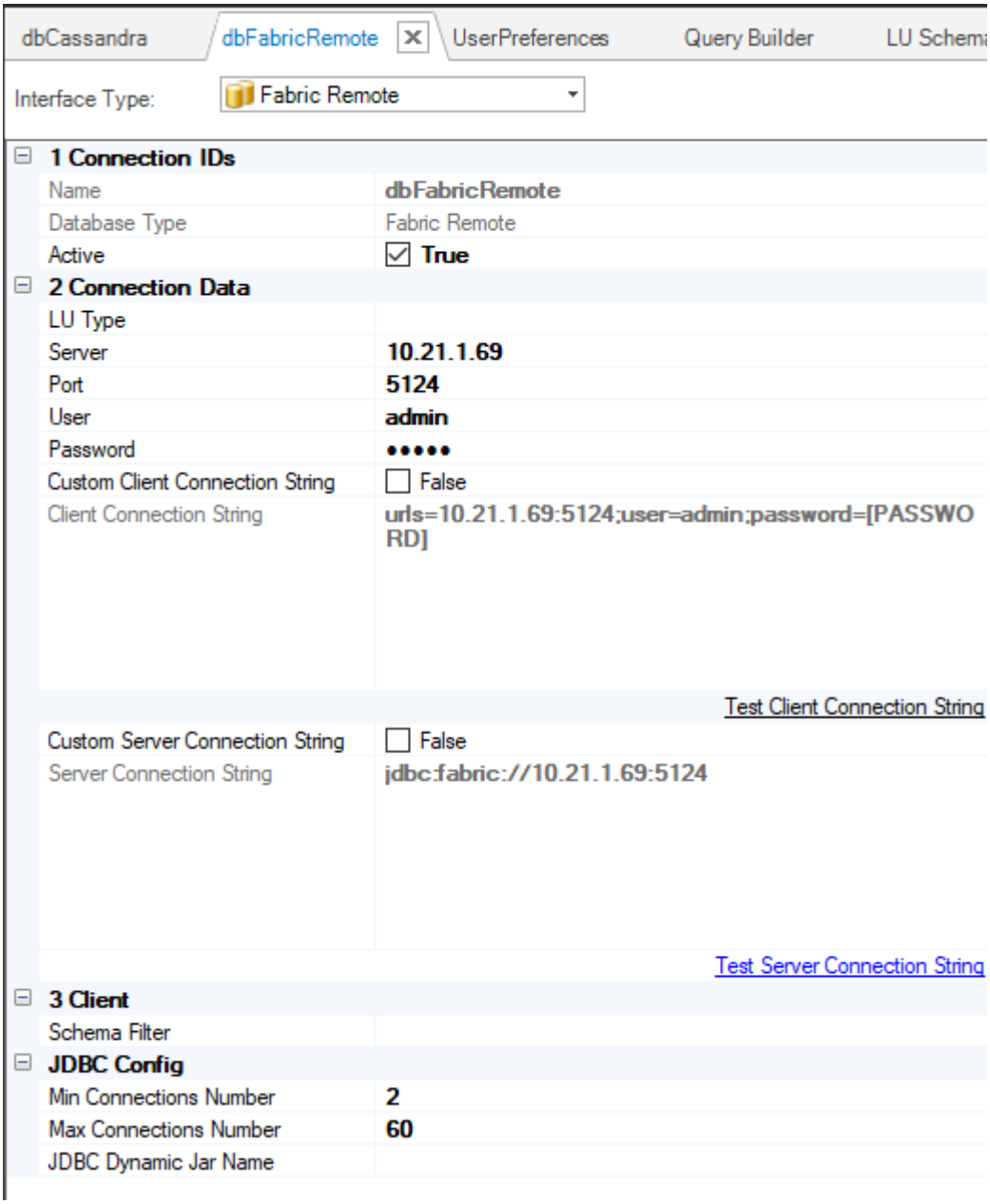
(2) Select Fabric Remote as **Interface Type** from the drop-down menu (at the top of the window).

(3) Set user and password for the internal Fabric interface.

(4) Set Server IP address.

(5) Click the **Save** icon ( ) or press CTRL+S to save the changes.

(6) From Fabric 5.0 and on it is required to define Fabric Remote in order to connect remote Fabric or in order to run a QueryBuilder on internal Fabric



(7) Before closing the new interface definition, verify that the connection parameters are properly set, by testing the connection (see Section 6.14).

6.14 Testing Interface Connection

☞ **To test the connection of the Interface:**

(1) Double-click the Interface name in the Project Tree pane.

(2) Click the **Test Client Connection String** link to test Studio and Query Builder connectivity.

(3) Click the **Test Client Connection String** link to test Server Side (Java) connectivity.

(4) A tooltip message appears displaying the results of the test, either a success or failure notification.

(5) Click **OK** in the dialog box.

6.15 Editing Interface Settings

☞ **To change the connection settings of a Data Source Interface:**

(1) Double-click the Interface name in the Project Tree pane.

(2) Edit the DB Connection parameters.

(3) Click the **Save icon (**  **)** or use CTRL+S to save your changes.

6.16 Deleting a Data Source Interface

☞ **To delete a defined Data Source Interface**

(1) Right click the name of the Interface (under Shared Objects/Databases) in the Project Tree.

(2) Select **Delete Selected Items** from the menu.

6.17 Clearing the Database Objects Cache

K2View Fabric Studio caches any changes to the database objects.

☞ **To clear the DB Objects Cache**

(1) Right click the name of the Interface (under Shared Objects/Databases) in the Project Tree.

(2) Select **Clear DB Objects Cache** from the menu.

6.18 Add SSL Support for Fabric and Cassandra Connections

- New 'Use SSL' checkbox was added to Cassandra/Fabric DB interface.. By default, this checkbox is cleared.
- When this check box is selected, it affects both the client and server connection strings (if they are not on custom mode):
  - For Fabric interface:
    - Client connection string- the following parameter is added to the connection string: **SSLMode=1;UseSslIdentityCheck=0;SSLTrustedCertsPath=C:\home\k2view\.cassandra_ssl\k2tls_CLIENT.cer.pem;**
    - Server connection string- the following parameter is added to the connection string: **&internodesssl=true**

> **Note**: for client connection string, you need to copy from your fabric server the following parameter: **$K2_HOME/.cassandra_ssl/cassandra.truststore** to **c:\k2view\cassandra_ssl** (create new directory) and populate this directory for **SSLTrustedCertsPath.**

  - For Cassandra interface:
    - Client connection string- the following parameter is added to the connection string: **SslProtocol=Tls12;**
    - Server connection string- the following parameter is added to the connection string: **&internodesssl=true**
- You need to edit Fabric Studio configuration file (k2FabricStudio.exe.config) to support data-viewer and debug over LU with Cassandra SSL connection:
  - Append 2 JVM args, trustStore and trustStorePassword to the new JavaArgs entry.
    The SSL arguments need to be added at the end of the entry value.

For example: look for “add key="JavaArgs”, and replace from the following input:

```
<add key="JavaArgs" value="-cp {0} -DuserLogFolder=&quot;{1}&quot; "/>
```

to the following value (path in the example bellow is provided only as an example). Save (in one line):

```
<add key="JavaArgs" value="-cp {0} -DuserLogFolder=&quot;{1}&quot; -Djavax.net.ssl.trustStore=c:/k2view/cassandra_ssl/cassandra.truststore -Djavax.net.ssl.trustStorePassword=<password>"/>
```

7 Working with Logical Units

7.1 Overview

The Logical Unit (LU) is a central concept in the K2View Fabric structure. The LU defines the structure of the data that is stored in K2View Fabric, based on mapping of the data that is stored in the source database.

The LU creates a hierarchical presentation of all the business entities involved in the project. The set of business entities is defined as part of the creation process that is described in this chapter. When selecting this set, it is important to properly analyze the expected use of the LU and characterize the business entities to be included.

For each LU you need to designate one of the business entities to be the single main business object. The remaining entities are considered to be the set of related subordinate entities. For example:

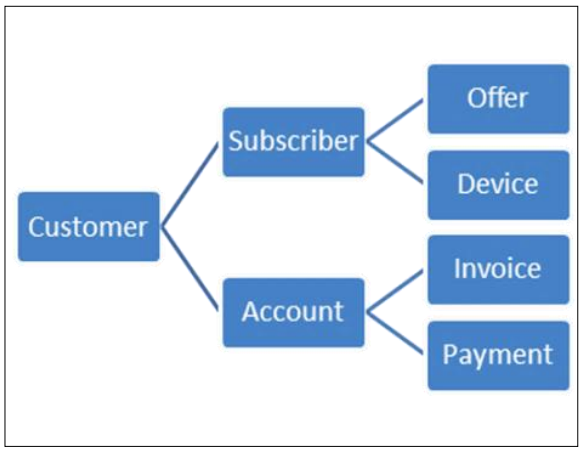


Figure 1: Logical Unit

In this example, the Main Object is the Customer, and the Subordinate Entities are the Subscriber, Account, Offer, Device, Invoice, and Payment. The entire complex is referred to as a Logical Unit (LU) called Customer.

At the next level in planning your LU, you need to select a single unique attribute of the main business object that identifies the individual instance. K2View Fabric handles the transformation of the entire Logical Unit data from the source database for a given instance into the K2View Fabric data structure.

7.2 Creating a Logical Unit - Overview

Creating a Logical Unit involves several steps, each a process in its own right. The following is an overview of these major steps.

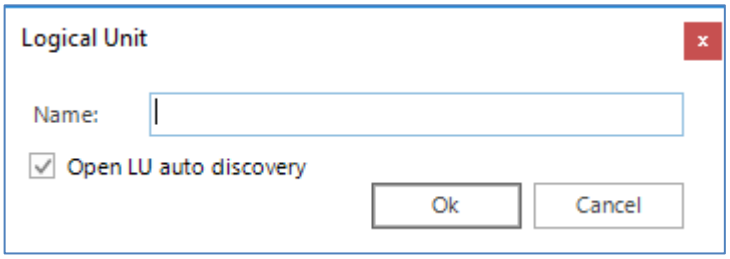
- Define the Logical Unit in the project structure.
- Select the main business object to include, as a table, in the Logical Unit.
- Select the additional source business entities to define as tables in the Logical Unit.
- Create the connections between the different tables.
- Add the transformations (functions, translations, globals) to the set of tables, see Chapter 9.
- Deploy the Logical Unit to the K2View Fabric Server, see Section7.13.
- Define the Instance Groups (List of instances) to be migrated from the source database to the K2View Fabric Data Base.

7.3 Defining a Logical Unit

☞ To define a new Logical Unit

- Right click the Logical Units in the project tree, and then select 'New Logical Unit' in the context menu.

  A Dialog box appears, allowing you to define the LU name.



- Enter the name of the Logical Unit in the Name field and click **Ok**.
- By default, the 'Open LU auto discovery' is checked. If this option is checked, the auto discovery wizard is automatically opened to create your LU. If you want to avoid using the auto wizard, you need to uncheck the 'Open LU auto discovery' option. For more information about the auto discovery wizard of the Logical Unit, see section 17.3.

  Your LU is added to your project tree.

7.4 Deleting a Logical Unit

☞ **To delete a Logical Unit.**

(1) Right click the Logical Unit in the Project Tree.

(2) Select **Delete Selected Items** from the context menu.

(3) Click **OK** in the dialog box.

7.5 Duplicating a Logical Unit

☞ **To duplicate a Logical Unit.**

(1) Right click the Logical Unit in the Project Tree.

(2) Select **Duplicate Logical Unit** from the menu.

(3) Enter a name for the new (duplicated) Logical Unit in the **Name** field,

(4) Click **OK**. (0)

7.6 Logical Unit Windows

When working with a specific Logical Unit, there are several windows that display the information for different views.

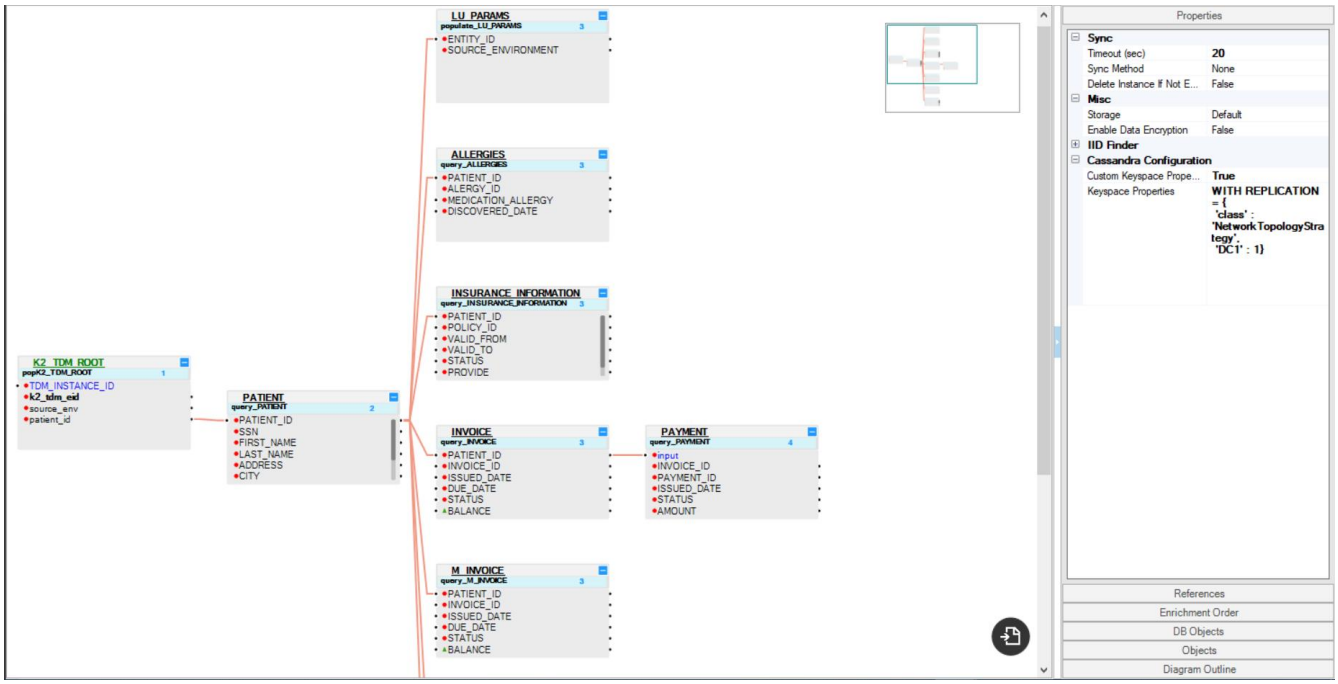
7.6.1 Logical Unit Schema Window

The **Logical Unit DB (LU)** window is a work area enabling users to define the Logical Unit structure.

To Open the Schema Window

(1) Double click the Schema node under the Logical Unit name in the project tree.

The Schema Window is displayed.



The main display presents the hierarchy of the tables included in the Logical Unit and the relationships between them (see above).

The right side of the window presents a panel that displays different information used to edit the Logical Unit. The information displayed is dependent upon the tabs that appear in the window. The tab that is placed directly above the display area determines the information displayed. The tabs are labeled:

Properties

Sync:

Sync is the mechanism that automatically synchronizes data between the source system and the K2View Fabric database. It retrieves the data from the source table and re-loads it into the table in the K2View Fabric database.

You can set the Sync properties on the LU, table, or table population levels.

- Sync property for a target object contains the following properties:

- Timeout (sec) – Set a timeout in seconds for syncing LUI.
- Sync method – None, Time inherited, and decision function.
- Perform sync every- set time interval for the sync.
- Delete an instance if does not exist- when True, an instance that does not exist in the source, when trying to synchronize – is deleted from the Fabric database. Otherwise, it is retained.

- **Storage-** enables setting the storage type of the LU and override the storage type, defined in the config.ini file. The following values are supported:
  - Default- the storage type is inherited from the config.ini file. This is the default value for the Storage property.
  - Cassandra- store the entity in the Cassandra.
  - None- the get entity always gets the instance from the source DB. The instance is not stored in the Cassandra.

> **Note-** you need to set the 'Storage' property to 'Cassandra' if the config.ini has a different Storage type, and you want to save the LU instances in the Cassandra. If the storage type of the config.ini is populated by 'Cassandra', you can populate the 'Storage' property by 'Default'.

- **Enable Data Encryption** – enables encrypting LU data in Fabric.
- IID Finder
  - Enabled - the default value is True
  - Source Schema - default LU Name
  - Delete Instance - True/False, in case of istance marked as black list.
- Key space
- **Cassandra Configuration**
  - Customer key spaces properties -should be set to True, if user wants to override default values set to the LU key space on the first deployment (for example setting RF to 3 on DC1 and RF 1 on DC3).
  - Key space properties - If the above field is set to True, should be populated with the required customized values.
- **References** – used to configure the Reference Tables that are accessible from this Logical Unit. For more information on using Reference Tables, see Section 16.
- **Enrichment Order** – used to define the order of the Enrichment Functions execution (see Section 9.2.7) by using the up and down arrows. Only enrichment functions that are directly related to specific tables are included in the display.
- **DB Objects** – displays the list of objects in the source database according to the Interface selected in the **DB Connection** field. DB objects can be refreshed by

clicking the Refresh icon that appears besides the DB Connection dropdown list. Dragging an object to the Diagram will create a new table in the schema.

- **Objects** – displays all table objects defined under the LU tree and are not present in the diagram already. Select and drag the required table into the Logical Unit diagram area to add it.
- **Diagram Outline** – displays an overview of the LU structure. Is used to search for a subset of tables included in the Schema.

7.6.2 Logical Unit Schema Window Tool Bar

By default, the LU Diagram window displays the following tool bar in the upper area of the window:

This toolbar includes the following icons:

 - Group Sub Graph

 - Ungroup Sub Graph

 - Update Tables from Database- when you click this icon, the schema of the LU tables is refreshed from the source DB. This action is needed if the schema of the LU tables was updated in the source DB.

> Note: update tables from DB does not add or delete tables, but only updates current tables. Add or delete tables from LU needs to be handled manually.

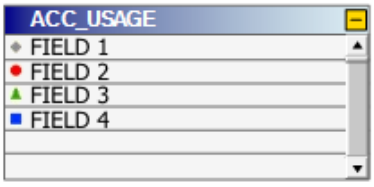
 - Refresh Items- Refreshes items from your implementation into the LU schema.

You can customize your window by adding and/or available toolbars for diagram windows. See more details in Section 3.7 of this document.

7.6.3 LU Diagram Work Area

When the main work area is in the LU Diagram tab, it displays the hierarchy of LU tables.

Each table is displayed with all of its fields in a scrollable table with the following features:



Each field is displayed in a separate line, by name and a small colored icon. The icon color indicates the data type of the field:

Grey: Blob

Red: Text

Green: Real

Blue: Integer

The main table is displayed on the left and the connections to the hierarchy are displayed as links between the tables.

The Schema Diagram work area is used to define the hierarchy of the tables included in the Logical Unit and the relationships between them.

> **Note:** Each Table Population object can be linked to a different parent table.

7.7 Adding Tables to a Schema

☞ **To add a source DB object into the Logical Unit**

(1) Open the **Schema Window**, as described in Section 7.6.1.

(2) Click the **DB Objects** tab in the right panel (see Section 8.6.1).

(3) Select the **DB Connection** to use to access the source DB.

(4) Select the object to add to the LU.

The object can either be dragged into the Schema Diagram work area, including all its columns, or can be created by dragging a subset of selected columns.

> **Note:** A Table object and a Table Population object are created automatically.

☞ **To add a fabric table into a Logical Unit**

(1) Create a K2View Fabric table as explained in Chapter 8.

(2) Open the **Schema Window** as described in Section 7.6.1.

(3) Click the **Objects** tab on the right panel.

(4) Select the object to add to the LU and drag into the Schema Diagram work area.

(5) Link it to the parent table.

7.7.1 Configuring the Instance PK Column

The root-table (or object) of the Logical Unit requires some additional processing after it is added into the Schema Diagram work area as explained above.

To configure the root-table

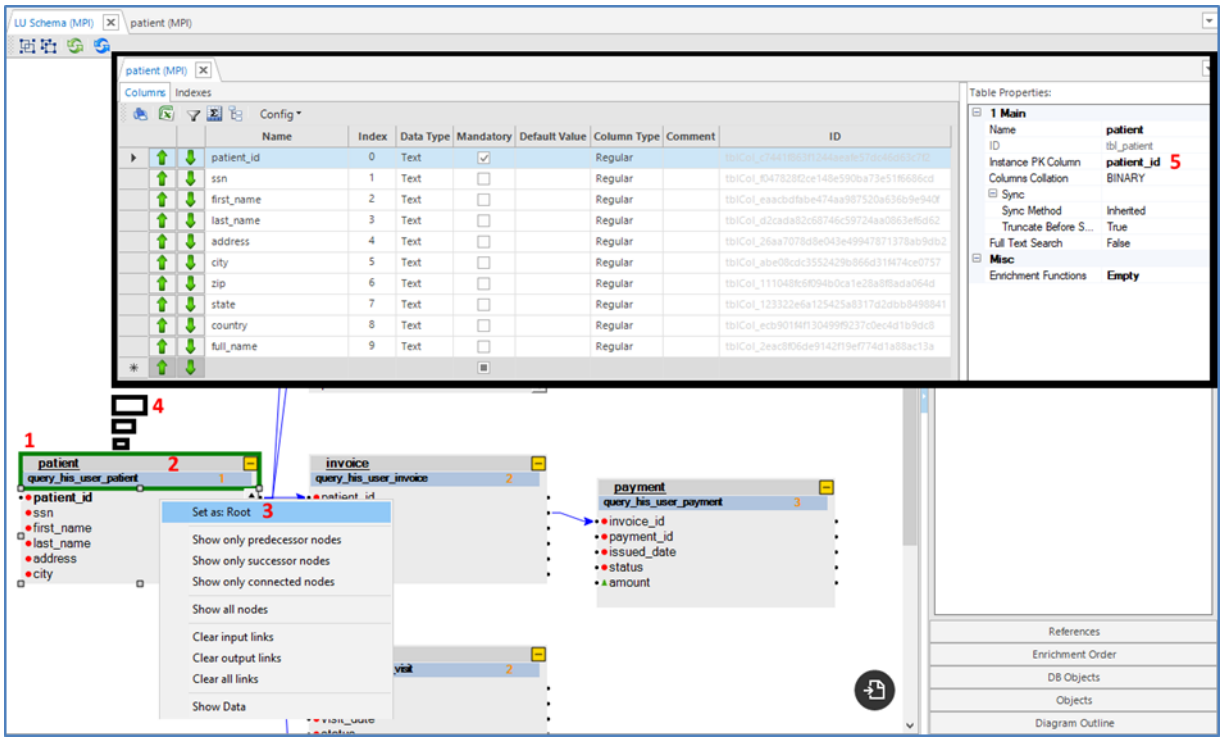
(1) Add the root-table to the **Schema Diagram** work area as explained above.

(2) Right-click the table header of the root-table in the **Schema Diagram** work area.

(3) Select **Set as: Root** from the menu.

(4) Double-click the table header of the root-table in the **Schema Diagram** work area.

(5) Select the table column that defines the main identifier of the instance in the table's Instance PK Column property.



7.7.2 Linking Table to Hierarchy

After dragging the root-table in the Schema Diagram, you can add additional entities into the LU hierarchy and link them together, creating the LU.

To add a table and link it to the hierarchy

(1) Add a table into the LU as explained in Section 8.2.

(2) Connect the table's population to the existing tables, based on the relation between them, by creating a link from an existing table.

> **Note:** If a table has more than one Table Population object, it is required to connect the input fields of each population to a parent table. The data for each table is fetched from the source system based on the parent columns connected to each Table Population object.

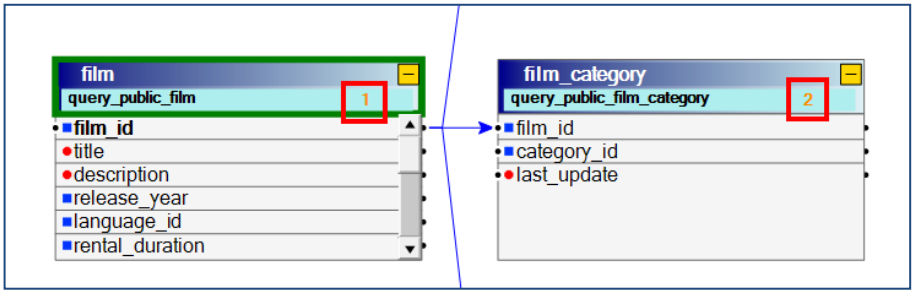
7.7.3 Define the Table Population Order

This feature enables you to define the table's execution order.

The order is defined on the right of the Table Population name.

The default order of a Table Population is set to the parent order plus one.

You can change the order by clicking it.



Note: Setting the order to a number which is smaller than the parent table order number is not allowed.

Reasons to utilize the order feature

If you want to create a new K2View Fabric table, **TABLE1**, which is based on data retrieved from two other tables, **TABLE2** and **TABLE3**, you must ensure that the **TABLE1 number** order is higher than the **TABLE2** and **TABLE3** order**.**

7.7.4 Disabling/Enabling Populations in the Schema

When necessary, you can add more than one population to a table. The disabling functionality enables you to disable a specific population, instead of having to delete it outright. You may want to do this, for example, for testing purposes.

Note: At least one population must be enabled in the root table.

To disable a population

In the *Schema* window, right-click the population row you want to disable and select **Disable Population**.

To enable a population

In the *Schema* window, right-click the population row you want to enable and select **Enable Population**.

7.8 Searching for a Specific Table in the Schema

☞ To search for tables in the Schema

(1) Open the Schema Window (see Section 7.6.1).

(2) Select the **Diagram Outline** tab in the right-side panel.

(3) Enter the name of the table of interest.

(4) Click the specific table of interest.

7.9 Clearing Links in a Logical Unit DB

☞ To clear Schema links for a table:

(1) Right-click one of the tables in the Schema display window.

(2) Select one of the options for clearing links:

Clear Link Options	
Clear input links	Remove all links to a parent table.
Clear output links	Remove all links to subordinate tables.
Clear all links	Remove all input and output links for the table.

7.10 Viewing Table Related Objects Only

When viewing the tables in the Schema window, you may want to focus on the immediate surroundings of a specific table only.

☞ To view the objects related to a specific table

(1) Right-click the table header in the Schema window.

(2) Select one of the options to view the tables directly related to this table. (See table below for the different options).

Menu Option	Functionality
Show only predecessor nodes	Show only nodes (tables) which are predecessor to this table
Show only successor nodes	Show only nodes (tables) which are successors of this table
Show only connected nodes	Show all nodes (tables) directly linked to this table

7.11 Viewing Table Data Directly from the Schema

Right click one of the nodes (tables) in the Schema and choose **Show Data.**

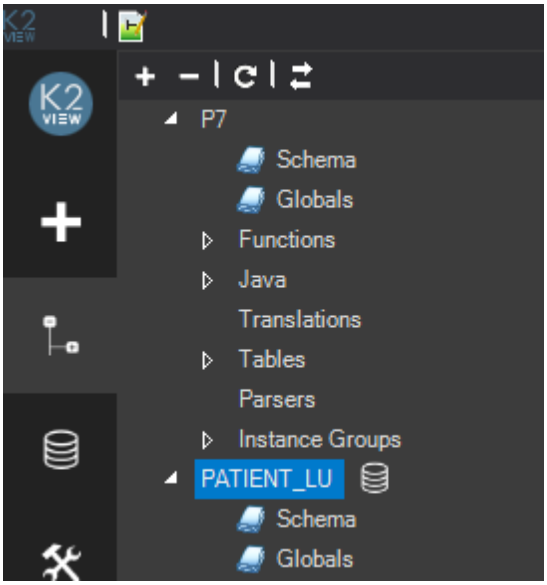
A query builder window opens automatically in the newest LU file.

7.12 Data Viewer Window

The Logical Unit DB Viewer tool enables users to view the Logical Unit database. Since the Logical Unit database is in the memory, it can be viewed by dumping the memory into a file. The purpose of the DB Viewer is to add debugging capabilities, improving the testing abilities and defect resolution process.

☞ To view the data stored in the Logical Unit

(1) Locate your mouse on your Logical Unit name in the project tree. An icon of data viewer will be displayed near the selected Logical Unit name. Then click this icon to open the data viewer for your Logical Unit.



A Data viewer window is displayed.

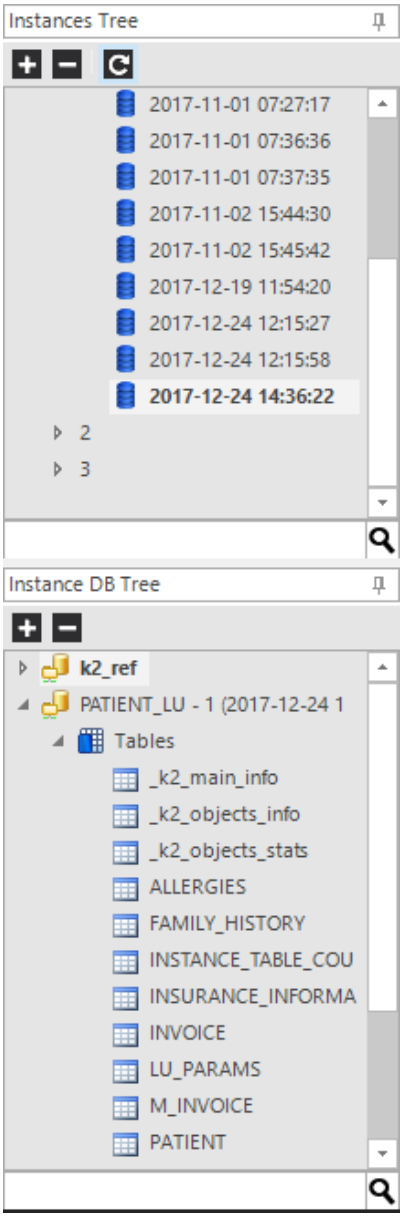
(2) Enter the Instance ID into the **Instance ID** field at the top of the central pane (instance ID or instance ID by function).

(3) Click the **Play** icon to the right of the field.

K2View Fabric Studio activates applications to retrieve data from source DB and activate all transformations defined within the LU. Upon completion, displays the tree of the generated LU instance file.

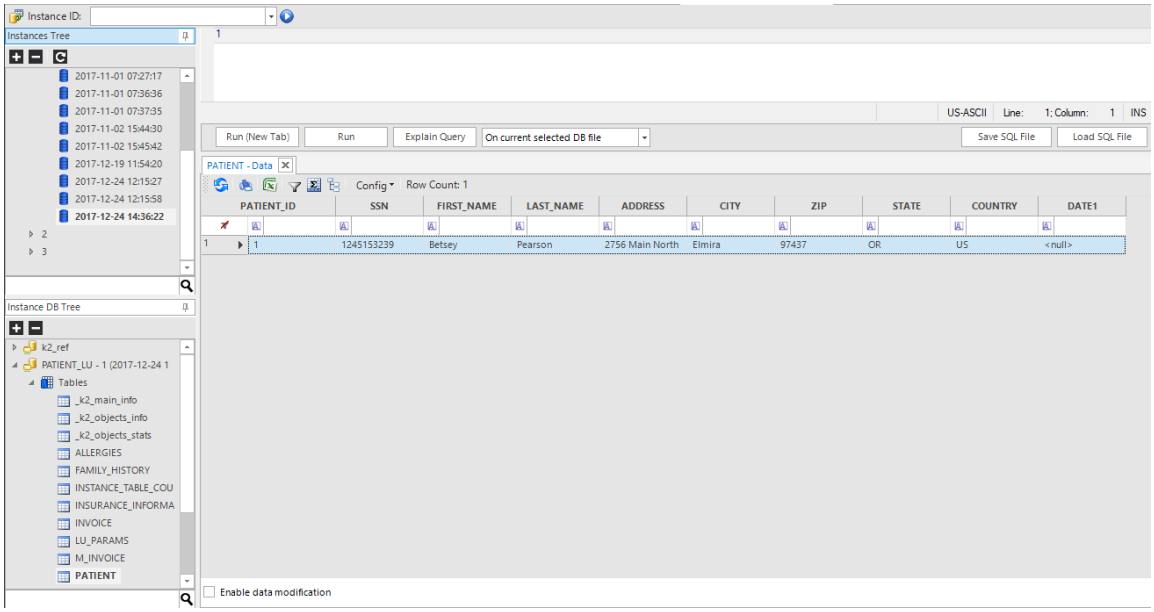
The data is retrieved, processed, and saved in a file that is labeled with the Instance ID and a time stamp of the creation date.

(4) Select a table from the **Tables** list of the DB instance tree:



Full data of table instance is displayed in the Data viewer window.

(5) The Logical Unit DB Viewer contains the following areas:



- **Import DB File** icon () - used to load and view an external data view file.
- **Instance ID** field is used to enter a specific instance value, or to select a previously stored instance ID (from pull-down menu), or use **Instance ID by function** - Fabric build-in function or a function written by implementer – function must return a string as output. Example for Instance ID by function: funcGetInst(234874).
- **Play** button () causes viewer to activate the process of retrieving and storing data file for supplied instance ID in the file system for debugging purpose.
- **Instances Tree** pane at the top left displays a tree of data files available for viewing.

  The instances in the tree appear according to the following order: LU, Instance ID, Dated file name.
- **Instance DB Tree** pane at the **bottom** left, displays the tree of tables in the K2View Fabric.

  The Tree includes the following product tables:
  - k2_lu_object_info – Contains statistics per table
  - k2_main_info – Contains the LU main information
  - k2_object_stats - Contains object timing statistics
  - Reference tables under k2_Ref

  Clicking any table in the tree displays the values for all columns.

  Right-clicking any table in the tree, provides the following options:

Option	Effect
Show Data	Displays the table or view in the Results pane
Show Schema	Displays the table structure in the Results pane
Show Indexes	Displays the table indexes in the Results pane

- The **Results** pane is in the bottom right side of the window.
  Displays the data or schema requested.

  Includes the following icons at the top of the pane:

Icon	Purpose
	Print results.
	Export results as an Excel file.
	Filter results by one or more columns.
	Toggle groupings.
	Toggle summaries.

- **Scripting** area in the upper right pane:
  An SQL scripting area where you can compose and run SQL statements in the selected Logical Unit DB.

  The area includes the following buttons at the bottom of the pane used to run the SQL statement.

- **Run** or **Run on New Tab**
  - Execute the given SQL statement.
  - Run on New Tab opens a new Results pane tab to display the results.
- Drop-down menu of special Run options:
  - **On current DB file**: The SQL is executed on the currently selected Instance File
  - **On newest DB file for each instance**: The SQL is executed on the newest Instance File of each instance in the instances tree
  - **On selected DB files**: The SQL is executed on the selected instance's files.

    Note: Use CTRL to select the required files.
  - **On all existing DB files**: The SQL is executed on all files in the instances tree.

> **Note:** If the user selects either 'On Newest DB file' for each object or 'On All Existing DB files', a new drop-down list, **Rows limit**, appears. Use this drop-down list to define the number of results to display.

- **Save SQL to File**: saves the current SQL statements to a file.
- **Load SQL from File**: retrieves an SQL statement from a file previously created in the Scripting area.

7.12.1 Running SQL Statement from Data Viewer

☞ **To run an SQL statement from the scripting area**

(1) Enter the SQL statement into the scripting area.

(2) Select an option from a drop-down menu (below the scripting area) to define where to run the statement.

(3) Click **Run** or Run on New Tab (below the scripting area).

(4) View results in the **Results** pane.

> **Note:** Instead of using the RUN buttons, an SQL statement can be executed using the F5 keyboard key or Ctrl + Enter.
>
> **Note:** The SQL statements must be separated by ';' in order to use these methods.

7.12.2 Exporting the Logical Unit Data File

☞ **To export Logical Unit Data file**

(1) Right click the required LU File in the Instance DB Tree pane.

(2) Select the **Export Selected DB Files** option.

(3) Select the location and file name to save the exported file.

(4) Click **Save** icon or use CTRL+S to save your changes.

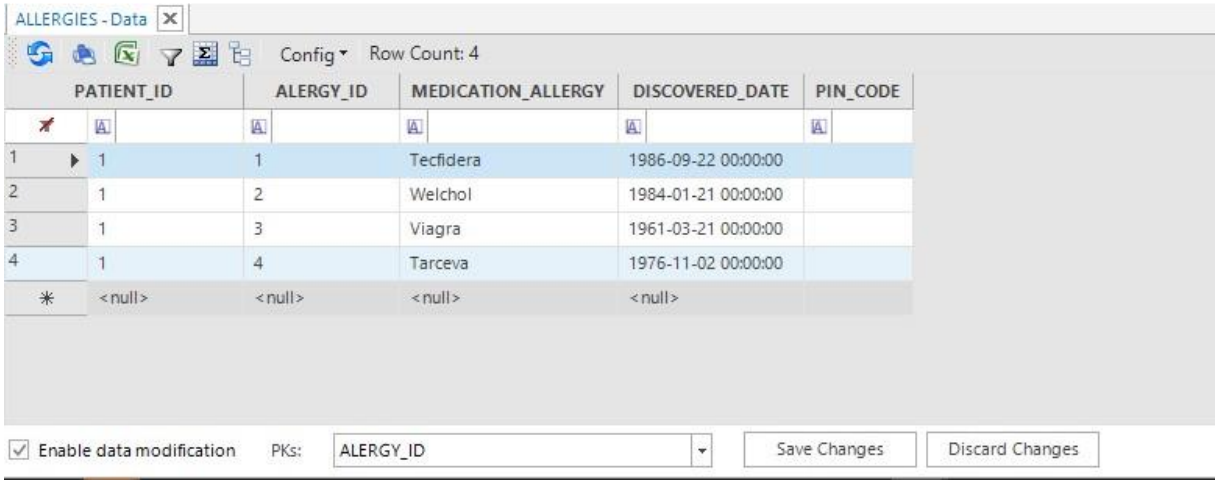
7.12.3 Updating the LU File

☞ **To update an LU data file**

(1) Double-click a table in the Instance DB Tree pane.

(2) Select the **Enable data modification** radio button displayed at the bottom of the Results pane.

(3) Select the PK to update in the Logical Unit table.



(4) Update the fields in the table for the rows to be updated.

(5) Click **Save Changes** button.

7.13 Deploying Logical Unit to Server

Logical Units created in the K2View Fabric Studio, or changes applied in the Studio are not reflected in the actual K2View Fabric DB until they are deployed to the K2View Fabric Server. The deployment includes definition of the schema and transformation rules such as functions, globals, and translations. However, it does not include migration of source data into K2View Fabric. To deploy the K2View Fabric project to the K2View Fabric DB Server:

(1) Verify that the target K2View Fabric Server is configured as described in Section 3.5.2.

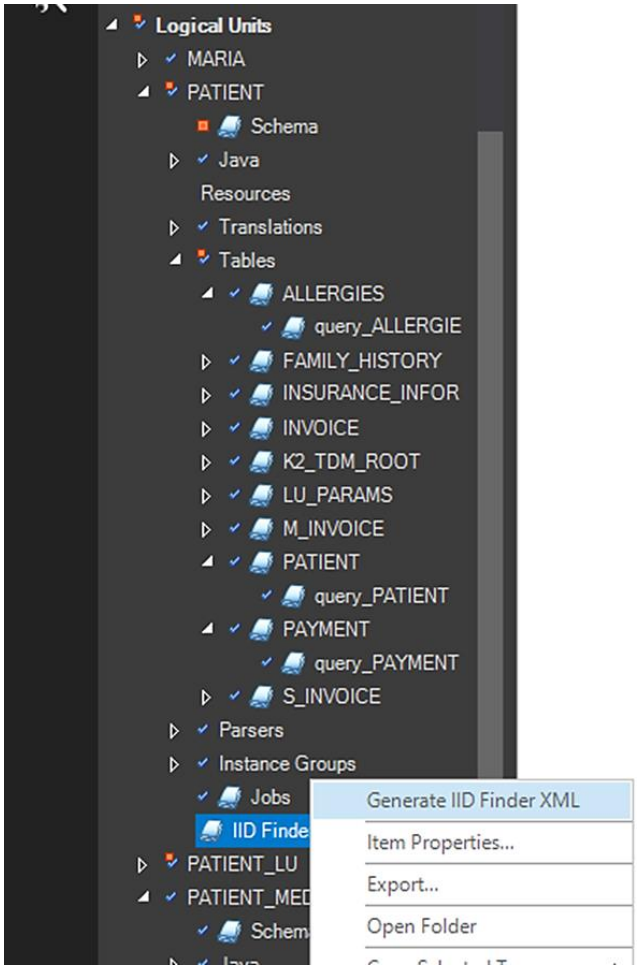
(2) Right-click the **Logical Unit** to deploy.

(3) Select the **Deploy to Server** option.

> **Note:** In release 5.3, if Logical Unit is marked as related to IIDFinder and IID Finder XML is generated, the staging.xml file will be copied to all Fabric servers as a part of the deploy process.

7.14 Generate IID Finder XML

(1) An option to generate staging.xml file for IID Finder functionality was added in release 5.3.



7.15 Defining Instance Groups

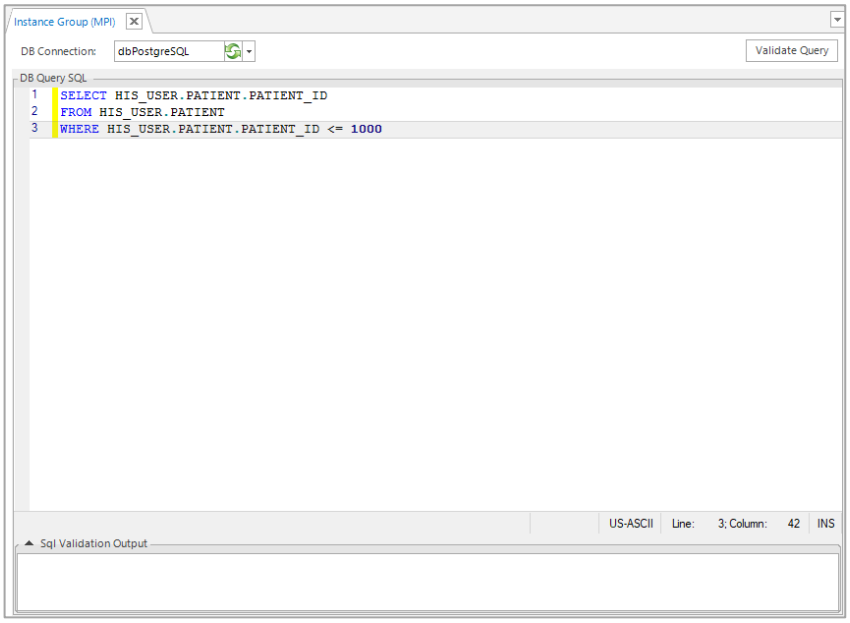
An instance group defines a population of instance IDs for migration and enables a migration of a partial population. The actual migration is performed through the K2View Fabric driver (refer to *K2View Fabric Runtime User Guide*).

☞ To define Instance Groups

(2) Right click the **Instance Groups** of the Logical Unit in the Project Tree panel.

(3) Select **New Instance Group** option.

The Instance Group Window is displayed.



(4) Select the DB Interface from the **DB Connection** drop-down list (in the window header area).

(5) Build the SQL Query that characterizes the Instance Group in the central working area.

(6) Click the **Validate Query** button (at the top right) to validate the syntax of the query.

(7) Click **Save** icon or use CTRL+S to save the definition into the Logical Unit.

(8) Enter name for the Instance Group and click OK.

> **Note**: Best practice is for Instance Group names to be prefixed with “ig”. The name should **not** include any special characters (for example, ‘#’ or ‘$’).
>
> **Note**: The syntax is automatically validated when the script is saved.

8 Working with Tables

8.1 Overview

The table is a basic building block for constructing the Logical Unit. In order to define a table, it is first required to define the table schema: columns, indexes, etc. The next step is to define one or more table population objects, which hold the transformation and mapping rules.

8.2 Define a New Table

The New Table window is used to define the table fields and their properties, indexes, and table properties.

☞ To open the New Table window

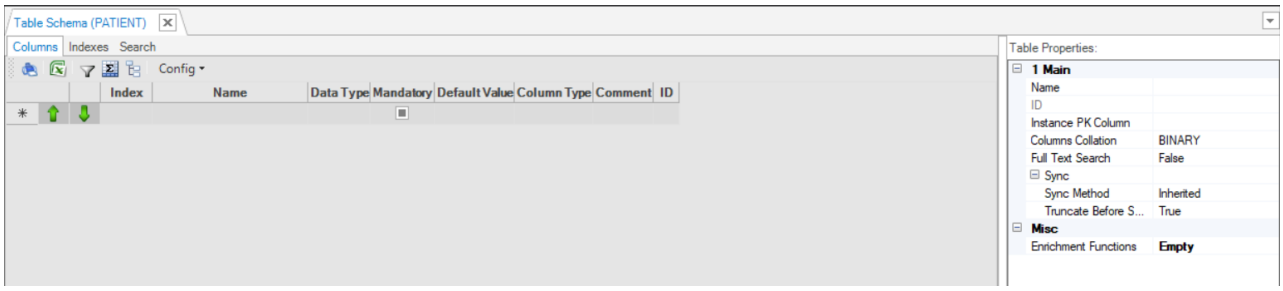
(1) From the **Implementation** tab, click **New Table** on the upper activity bar and select the relevant LU.

(2) Go to **Logical Unit** in the project tree, right click **Table**, and select **New Table** from the pop-up menu.

(3) From the **Schema** window, drag a database table from the DB objects tab.

> **Note:** This is the **simplest** way to create a table and a table population in one step.

The New Table window is displayed.



8.2.1 Define Table Columns

Add column to a table → In the **Columns** tab, populate a new record as follows:

- **Name** – name of the column
- **Index** - Defines the location of the column. It can be changed in order to move up or down the column in the table structure.
- **Data Type** - The data type of the values in the column. The following types are supported:
  - Integer
  - Real
  - Text
  - Blob
- **Mandatory** – Select the checkbox in case this column must be populated (not null), otherwise leave it clear.
- **Default Value** - The value set in this column is used while populating the table in case no input link exists for this column.
- **Column Type** - Two column types are supported:
  - Regular – (default) Column populated using the table mapping.
  - Computed Field – This means that the column is updated using the enrichment function as explained in Section 9.2.7.

The difference between those two types is the execution timing. When a column is defined as a computed column, the execution of the function is performed after all tables in a Logical Unit were already populated.

- **Comment** - You can add a comment for a specific column.
- **ID** – A unique ID of the column generated by the K2View Fabric Studio.

8.2.2 Delete Columns

Right-click and select either Remove All Columns or Remove Selected Columns to delete specific columns.

8.2.3 Define Table Indexes

Indexes are being used in order to improve the selection from the K2View Fabric tables.

☞ To add an index to a specific column/s

(1) In the column tab, select the required column row.

(2) Right-click and select **Create Index From Selected Columns**.

> **Note:** You can select more than one column in order to create an index with multiple columns.

☞ To update or delete an index

In the indexes tab:

(a) Click **'X'** to remove the index.

(b) Edit index properties:

The following field's checkbox can be marked/cleared:

Instance only	Selected – indicates that the index affects a single instance only. Cleared – indicates that the index affects also while performing cross-instance queries.
PK	Selected – indicates that an index is created as a Primary Key.
Unique	Selected – indicates that the column is unique.

8.2.4 Define Search Fields for Table

Fabric enables defining list of fields, available for cross instance search. For example, if you need to enable a search of all customers having the same first name and last name, you need to define first name and last name fields of the customer schema as search fields.

☞ To add a search index to a specific column/s

(1) On the column tab, select the required column row.

(2) Right-click and select '**Create Search Index From Selected Columns'**.

(3) Select the 'keyword','date' or 'data' options.
Keyword field enables a search by this field.

To enable a search on a date field, set its type to 'date'.

Data field can be returned by the search, but you cannot initiate a search by this field.

(4) Example:

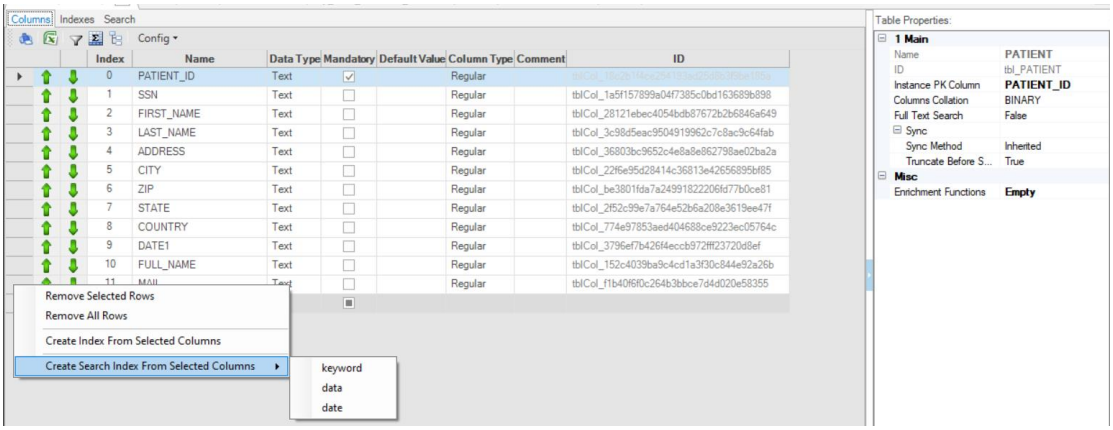
(5) First Name and Last Name are keyword fields.

(6) Customer id is a data field.

(7) You can run a search to get the list of all customer IDs called "John Doe".

Notes:

- You must create a Search Engine interface and populate it with the connection details of the Elasticsearch for your search. This interface is needed to enable Fabric connecting the search engine when running the search command.
- You can select more than one column in order to create search index with multiple columns.
- You can define several indexes on one column.
- The ElasticSearch engine must be up and running to deploy your LU when your implementation contains a Search Engine interface.
- Date fields must be assigned a date format, identified by the Elastic Search. See the list of the date formats, supported by the Elastic Search: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-date-format.html#built-in-date-formats



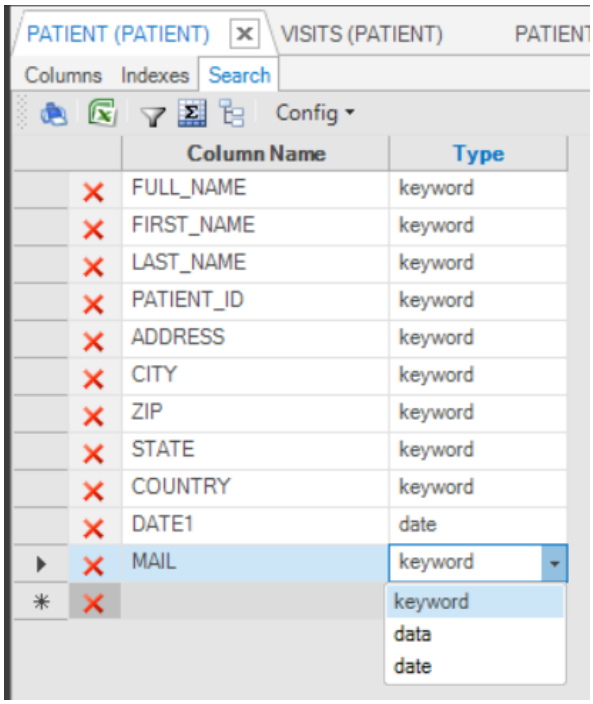
To update or delete search index

On the search tab:

(1) Click **'X'** to remove the search index.

(2) Edit index properties by selecting one of the available types: .

(3)



(0)

(a)

> **Note:** You must define the fields required for search, before you deploy the LU. If you wish to add/remove/update search fields after the deploy of the LU, you must drop the LU before you re-deploy it.

8.2.5 Custom Data Change Fields

Fabric has a full CDC (Change Data Capture) solution to notify external system on data changes, published through Kafka topic.

You can customize Fabric Studio and add consumers for the CDC fields by editing the <project name>.proj field of your Fabric project:

(1) Right-click the project name and select the 'Open Folder' option.

(2) Open the <project name>.proj file for editing.

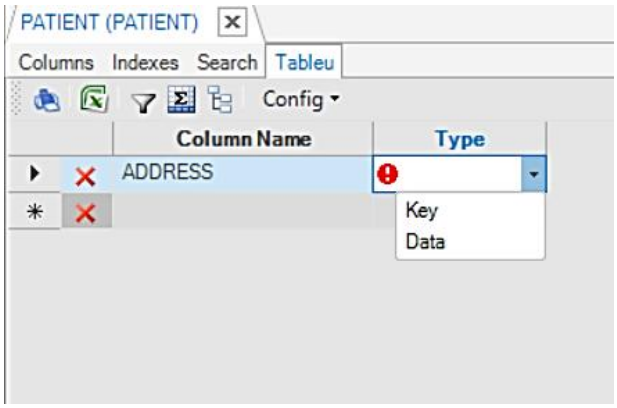
(3) Edit the DataChangeIndicators tag- add 'DataChange' tag. See the example below:

```
<DataChange name="Tableu" enabled="true">
  <Options>
   <option>Key</option>
   <option>Data</option>
  </Options>
 </DataChange>
```

(4) Save and close the .proj file.

(5) Close and reopen your project to reload the changes of the .proj file.

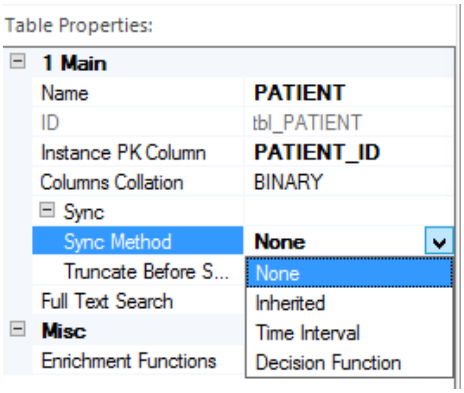
(6) Open the required LU table to be the new tab, added for data change. See the example below:



8.2.6 Table Properties

The following are the properties for LU tables:

Property	Description
Name	Name of the table

Property	Description
ID	Internal ID of the table object
Instance PK column	A unique field that is used as the Logical Unit instance main identifier
Columns Collation	> BINARY- compares the exact string in the field with the SQL statement. This is the default option. > NOCASE- allows the select statement to ignore upper/lower case when comparing text fields. Example: "Select TYPE from tblExample where NAME = 'value'" returns records where NAME field is set either to 'VALUE' or 'value'. > RTRIM - allows the select statement to ignore white space characters from the right hand of the string when comparing text fields. Example: "Select TYPE from tblExample where NAME = 'value'" returns records that match both 'value' and 'value '.
Sync	**Sync-** sync is the mechanism, which automatically synchronizes data between the source system and the K2View Fabric database. It retrieves the data from the source table and re-loads it into the table in the K2View Fabric database. Sync property for a target object contains the following properties:  **None** - no Sync. **Inherited** - a default value. Inherits the sync method from the schema level. The Inherited mode is available only in the Table and Table population. When the Inherited mode is selected, the sync method is based on the object parent, table population from the table, and table from schema.

Property	Description
	Time Interval - defines the time interval in which the table's data needs to be synchronized. The format is D.HH:DD:MM and the default is 1 day. The time interval is checked by K2View Fabric whenever the user tries to access data from the K2View Fabric for an instance. It checks if the latest update of data in K2View Fabric occurred before the time interval for the specific instance. If yes, data is extracted from the source and re-loaded to the K2View Fabric data base. Otherwise, the existing data in K2View Fabric is being used. For example- if a user sets 'Perform Sync Every' to 01:00:00, and retrieves data from K2View Fabric as 9:00 AM, then K2View Fabric checks what is the latest update date for this instance and the table. If the latest data was loaded before 8:00 AM, K2View Fabric extracts data for the table from the source system and re-loads it to K2View Fabric. **Decision Function**- a user function that the user can decide by java code if sync will occur using built-in functions –getMigrationID , getLastSyncTime, getTableName , getPopulationName. The sync is performed based on this code. When Decision Function is selected, a user needs to choose a function from a drop-down list of the available decision functions. **Truncate Before Sync** – default is **True**. This means that the data, which already exists in K2View Fabric for this table is truncated within the synchronization process and reloaded from the source. In case this property is set to **False**, the data from the source is appended on top of the data, which already exists in the table. Truncate before sync can be set at the LU, table, and population levels.
Full Text Search	The user must set the full text search to **True** to enable the use of **MATCH** sqlite command as part of the WHERE clause of a SELECT statement that reads data from the K2View Fabric table. Default is **False**. See more information about the **MATCH** command in the following link: http://www.sqlite.org/fts3.html#section_3
Enrichment Functions	■ Attached functions, which are executed after all LU tables are populated. ■ The Execution order is determined on the LU level. ■ The table sync property is inherited by those functions. ■ The incentive to attach function to a table is when the function updates (Computed field) /deletes/inserts additional records to a table. ■ When clicking the 3 dots, a window, which enables you to choose the enrichment functions to be attached, pops up. Only functions without input\output parameters appear in the drop-down list.

8.3 Table Population Window

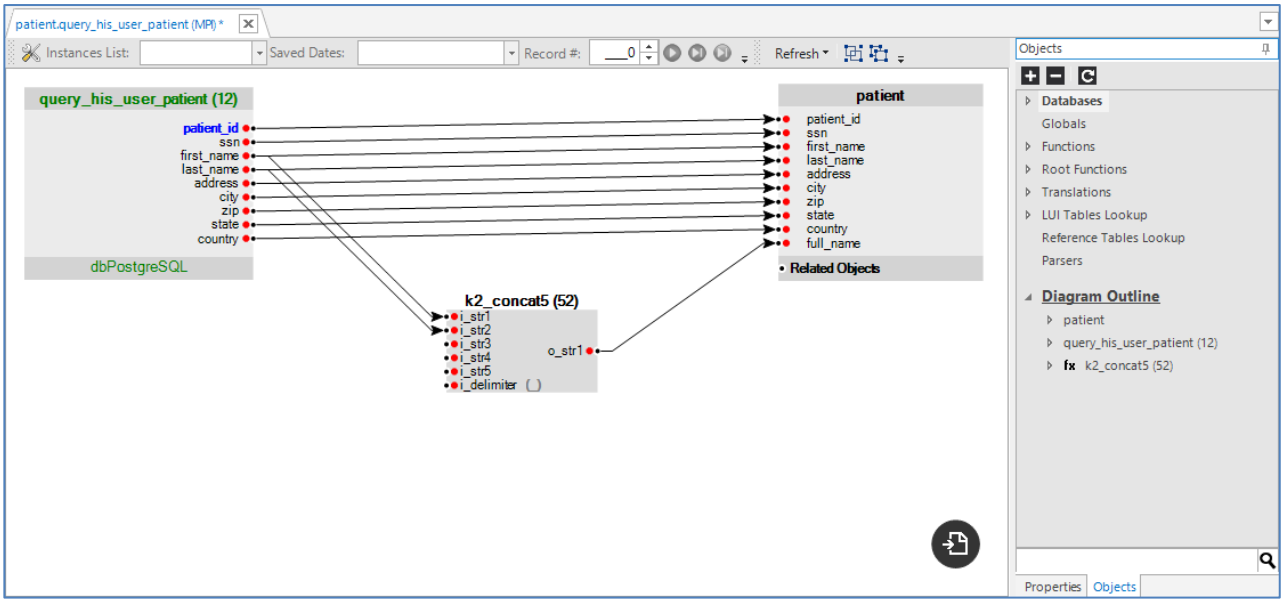
The Table Population window is used to define the transformations applied to DB table data when loading to the K2View Fabric database. It provides a clear and visual display of the rules.

- Left panel: Input
- Middle area: Data transformations
- Right panel: Output

To open the Table Population window

Go to **Logical Unit** in the project tree, double click one of the **Table Population** objects.

The Table Population window is displayed.



The Table Population window is comprised of the following parts:

- Window header – contains the debug toolbar:

	Enable/Disable Debug
	Select instance to debug or a enter new instance
	Select saved file or generate new
	Display the current record in process
	Debug

	Debug Next Record
	Debug Next instance

> Note: You can customize your header and add additional toolbars. For more details see Section 3.7

- Working area. The working area is the workspace for mapping the links, translations, and functions between the source and the K2View Fabric tables. The display shows:
  - Query table on the left – the table from the source DB.
  - K2View Fabric table on the right – the table in the Schema.
  - Mapping area – Links, Functions, and Translations that represent the data transformation from the source to K2View Fabric columns.
- Properties tab at the upper right side of the window

  This area shows the properties for each selected object in the Table Population object (Table, LU Lookup, function, Translation), some of which are editable. Each object's properties are described in the "Table Properties" Section 8.2.4.
- Objects tab at the right side of the window:
  - Objects: enables users to select objects, where the object is a Translation, function, Global, LU Lookup, or a source table.
  - Diagram Outline: outline of the elements within the Table Population object by type or by group
  - Refresh Diagram Outline and Object trees:

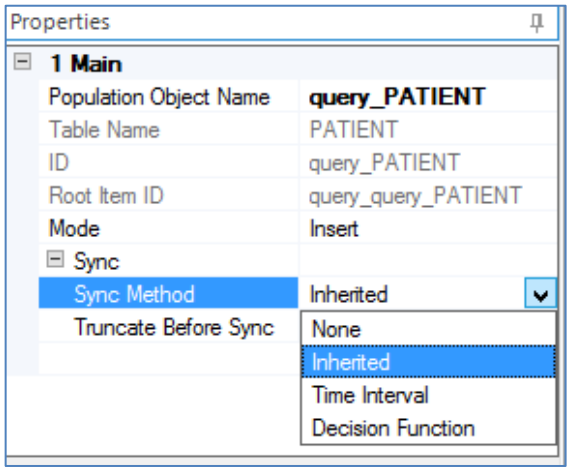
    Click the Refresh icon ().

☞ Set sync method on population level

Sync method can be set also on population level.

Create new table population and select sync method.

- None
- **Inherited** (By default)
- Time Interval
- Decision Function



> **Note:** Refer to Section 8.2 for more details of all sync methods.

☞ Set Truncate Before Sync on population level

If you set Truncate Before Sync to true, the **entire LUDB table** is truncated before the population whenever the population is executed.

> **Note:** If you set a different sync method for each population, and you want to avoid a truncate of an entire table whenever one of the populations syncs the table, set the mode of your populations to 'Upsert' and set Truncate Before Sync property to 'False' for your population objects.

☞ Set IID Finder properties on population level

Set the IID Finder properties on the population level to be used when creating the staging.xml file.

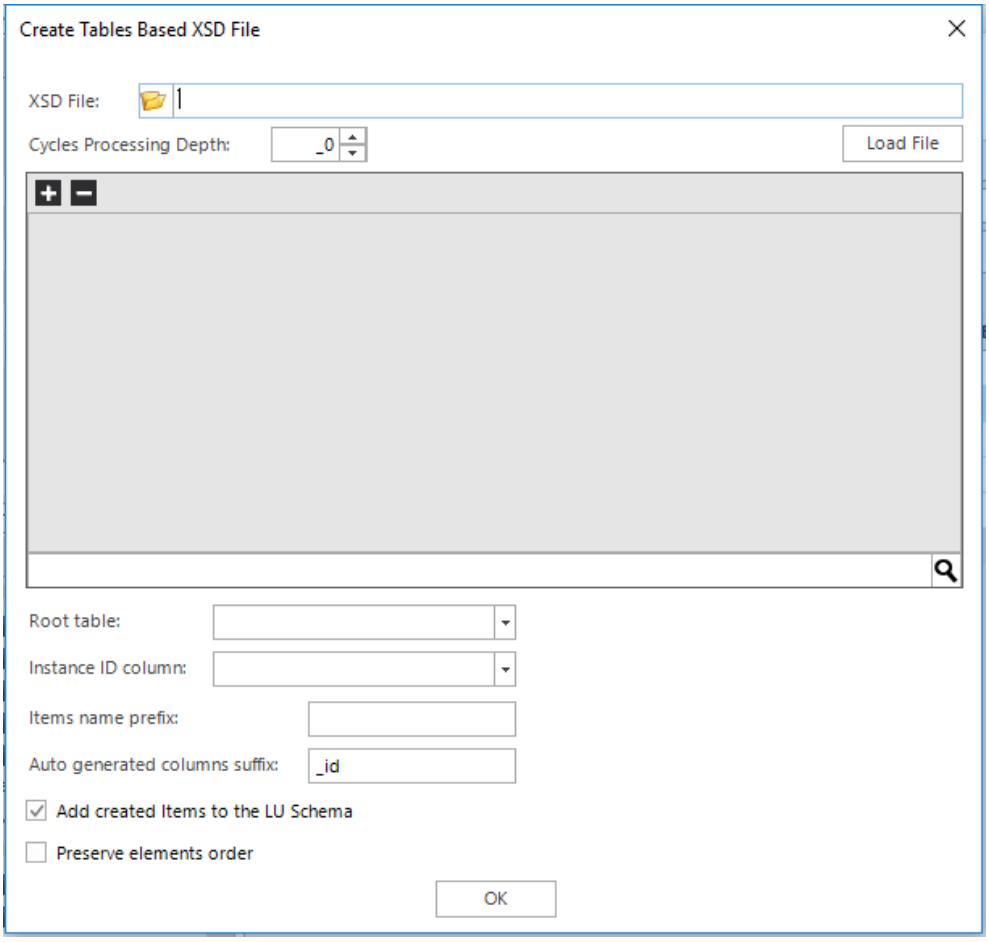
8.4 Creating a Table from an XSD File

This functionality enables you to create a set of tables from a given XSD. Each table represents a complex element of the XML.

☞ To automatically generate tables from an XSD file:

(1) Under the table object, right-click and select **Generate Tables Based XSD**.

The following window displays:



All tables and LU schema tabs must be closed before performing this activity. If one of these tabs is open, you are prompted automatically to confirm that it is closed.

(2) Click the **Directory** button at the upper left of the window. A Windows directory window opens.

(3) Select the required XSD file and click **Open**.

(4) Set the **Cycles Processing Depth** value. The default value is zero. This value will only be updated if the XSD element types have cycle references.

(5) Click the **Load File** button. The schema elements of the file are displayed.

(6) Select the required elements to be mapped and placed in the XML by selecting the appropriate checkboxes and setting the values for the following fields:

- **Items Name Prefix:** The name is set according to the XSD element name with this field as the prefix. The default is Table.
- **Auto Generated Columns Suffix:** In this field, specify the suffix to be assigned to the column name.
- **Add created items to the LU Schema:** If this checkbox is checked, all tables are automatically added to the LU schema as a group. The tables are linked to each other based on the relationships defined in the XSD. When this checkbox is not checked, the items are not part of the runtime environment.
- **Preserve elements order:** If this checkbox is checked, the order of the XML elements matches the order in the XSD file.

The **Preserve elements** order field is cleared by default. It is recommended to select it only if it is of an absolute necessity.

(7) Click **OK**.

The LU schema opens automatically. If the **Add created items to the LU schema** checkbox was checked. The schema includes a subgraph that has all the complex element tables created by this activity.

- Tables created for each complex element are marked as follows:
  - For each simple element or attribute, a table field is generated under the parent element.
  - Fields with the **k2id** postfix are generated automatically by the tool. These fields link the tables based on the relationships defined in the XSD.

K2id fields must be populated in order to establish the correct links between the XML elements.

(8) Connect the root element of the XSD to one of the main LU schema objects and click the **Save** icon.

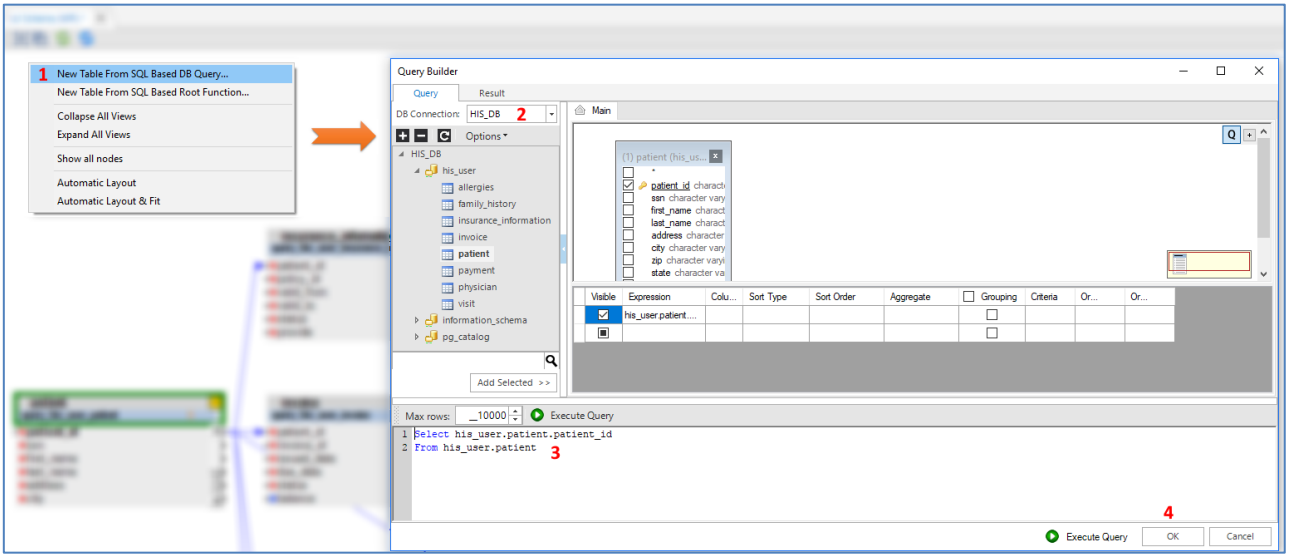
8.5 Creating a Table Based on a Database Query

This functionality enables you to create a table based on a database query.

To create a table based on a database

(1) In the Schema window, right-click in the middle of the window and select the **New Table Based on DB Query** option.

The Query Builder window displays:



(2) In the **DB Connection** field, select the connection on which the table is to be based.

(3) Define the query on which to base the table. All other aspects of using the query builder are as described in the section *17.2*.

(4) Click **OK**.

(5) In the *Table Name* window, specify the name assigned to the table.

(6) Connect this table to its parent table.

8.6 Working with Input Settings

8.6.1 Creating and Defining a Table Population Object

Go to the **Logical Unit** in the project tree, right click a specific **Table**, and select **New Table Population** from the pop-up menu.

The first step in defining the Table Population object is creating the root input object. Select the Objects tab on the right side of the screen. A Table object can be populated by the following root input objects: Databases or Root functions. The root object determines the number of records to be inserted into the table.

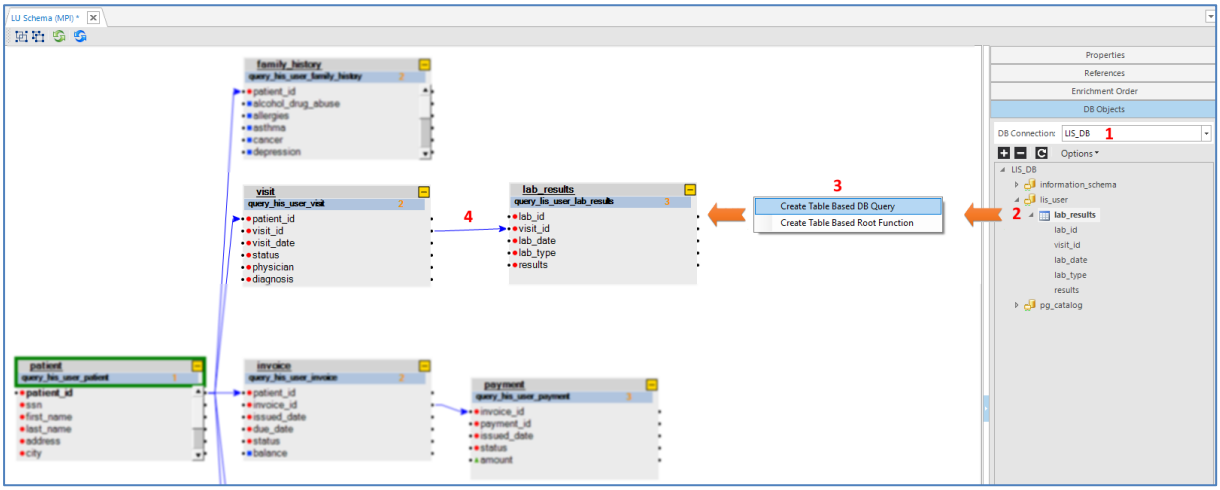
8.6.1.1 Databases

(1) Open the **DB Objects** option in the object tab on the right side of the screen below and select the relevant database.

(2) Navigate and drag the requested table into the working area.

(3) Select **Create as DB Query**.

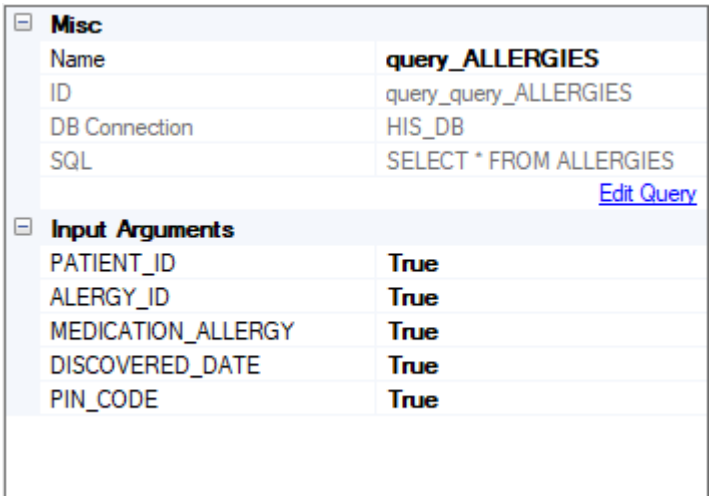
(4) Link the Table to a parent.



> **Note:** The root input table must be defined as DB query. DB query may populate the table object by many records.
>
> **Note:** Only one DB Query object can be defined for a table object.

(5) If there is a need to retrieve data from several input tables, use **Edit SQL query** to have a join of several input tables. See explanation below.

(6) Edit SQL query for the table query → Double click the input table query (left side) or click the **Edit Query** option in the properties window on the right side of the window.



In the query builder window, you can choose specific columns and conditions to build the SQL query for the table (as described in the Query Builder Section 17.2). Then click the OK button. See the screen below:

8.6.1.2 Root Functions

Open the **Root Functions** option in the object tab on the right side of the screen above and drag the required function into the working area.

Each record returned from the function, is inserted into the table.

The output fields, returned by the function, can be translated as mapped, as explained below.

Another option is dragging the required table from the objects tab, and then selecting the "Create as Root Function" option. A root function will be generated automatically.

> **Note:** For detailed explanation on how to write a root function go to **Root Functions,** Section 9.2.6.

8.7 Working with Transformation Settings

8.7.1 Adding a Table Lookup

A lookup object is added to a table to get some additional information from source DBs, other LU tables, or from a reference table for each record, retrieved by the root object. Lookup returns only one record for each input record. For example, you may need to get all bills of a customer, and, add the billing address for each bill record. In this case, the address table is defined as a lookup table to be connected with the bill table.

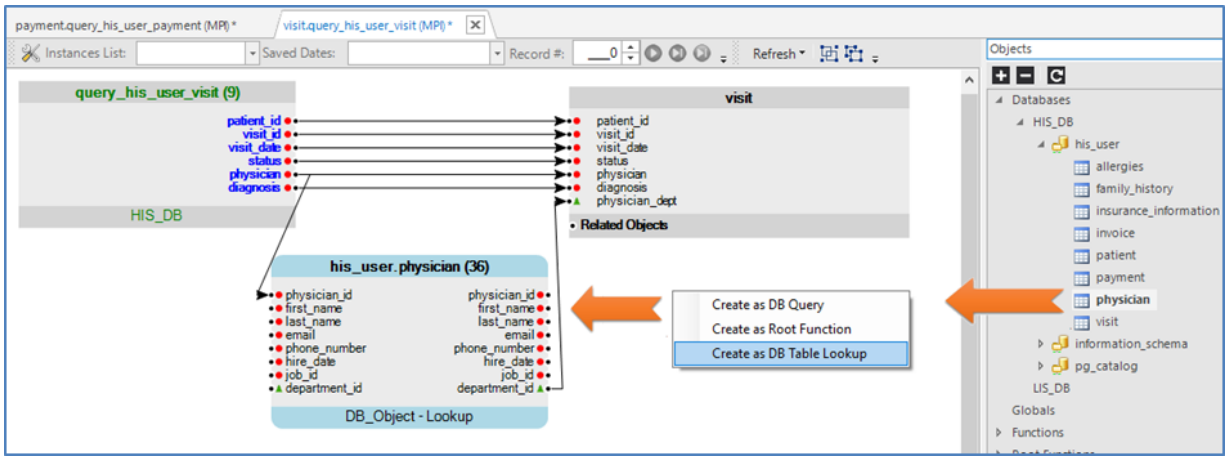
☞ To add a DB table as a Table Lookup

(1) Open the **Table Population** window (see Section 8.3).

(2) Select the **Objects** tab in the lower right panel of the mapping window.

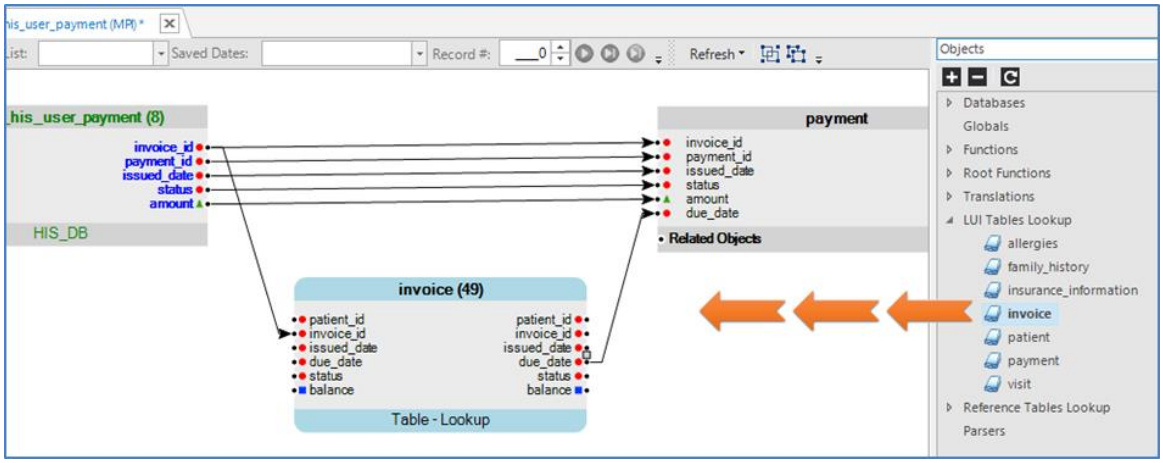
(3) Database Lookup:

Select the requested table from the Objects pane -> Databases -> Click **Create as DB Table Lookup** and link the input and output of the lookup.



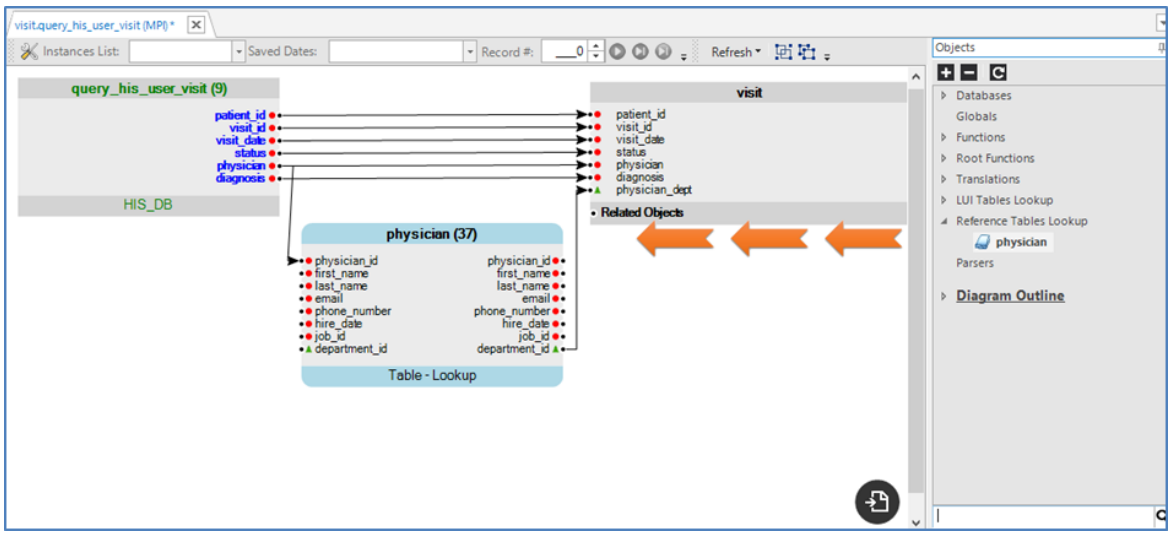
(4) For LUI Tables Lookup:

Select LUI Tables Lookup in the Objects tree, Select the relevant LU table and drag it into the working area.



(5) For Reference Tables Lookup:

Select Reference Tables Lookup in the Objects tree, Select the relevant reference table and drag it into the working area.



A Table (that was dragged into the working area) is displayed as a table lookup object.

(6) Drag input lines from the DB mapped table into the Table Lookup object.

(7) Drag output lines from the Table Lookup object to the LU table at the proper receiving columns.

The corresponding SQL query is generated for the table lookup.

> **Note:** The input lines are used for the 'where' part statement, the output lines are used for the 'select' part of the statement.
>
> **Note:** If you create a lookup on the LU table, you need to make sure that the lookup object is being populated prior to the current table. Meaning, this table will have a higher order execution number than the lookup object.

8.7.2 Adding a Table based on Root Function

(1) Open the **Table Population** window (see Section 8.3).

(2) Select the **Objects** tab in the lower right panel of the mapping window.

(3) Select the **Databases** item in the Objects tree.

(4) Select a relevant interface from the list of DB Interfaces.

(5) Drag the required table into the working area.

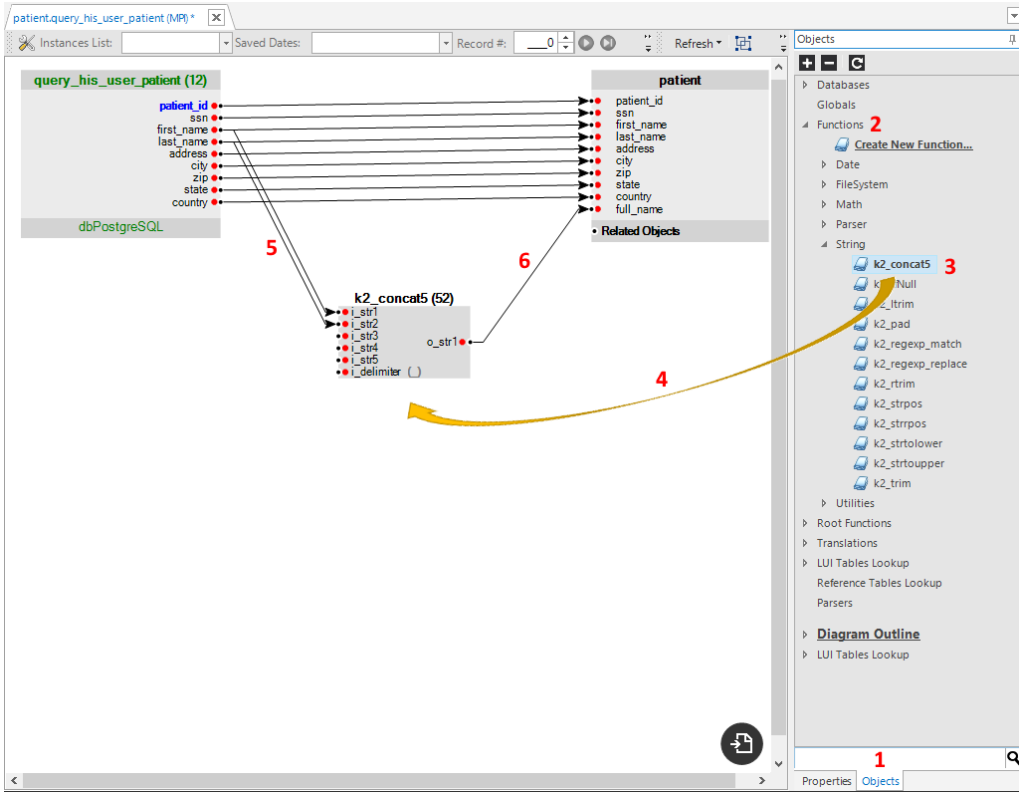
*The Context menu displayed in the working area, with 3 options – **Create as DB Query, Create as DB Table Lookup,** and create as root function.*

(6) Click **Create as Root Function.**

8.7.3 Adding Functions

Functions are used to transform data from the source DB and produce new information for the LU objects (for more information on Functions see Section 9.2). K2View Fabric Studio supplies a basic set of functions that can be used to perform many essential transformations. For more information on the parameters and transformations of these functions, see Section 9.2.

Adding a Function to the table population map



(1) Select the **Objects** tab in the right panel of the mapping window.
(2) Click the **Functions** item.
(3) Select the required function from the functions list.
(4) Drag the function into the working area.
(5) Connect input links from the source columns to the function input parameters.
(6) Connect output links from the function output parameters to the LU columns.

☞ To add a new Function to the table mapping

There are 2 ways to add a new function from the table population window:

(1) Right click anywhere in the Map. A pop-up menu appears.
(2) Then select **Insert New Function.**
(3) Click on **Create New Function** option under Functions in the object tab.
A new function window is opened.
(4) Define the function as described in Section 9.2 and click the '**Save** icon or press CTRL+S.
The function is added automatically to the table mapping window.
(5) Continue with step 5 as mentioned above.

> **Note:** There is no limitation on the number of functions that can be connected in serial or in parallel. There is no limitation on the combination of Translations and functions.
>
> **Note:** Root function cannot be used in the mapping area (Those functions can only be used in the input area mentioned above).

8.7.4 Adding Translation Object

You can add any translation, which is defined under the specific LU or under shared objects (see Section 9.3). The translation is executed for each record of the table population query.

☞ To add an existing Translation to the Table Population object

(1) Select the **Objects** tab in the in the right panel of the mapping window.

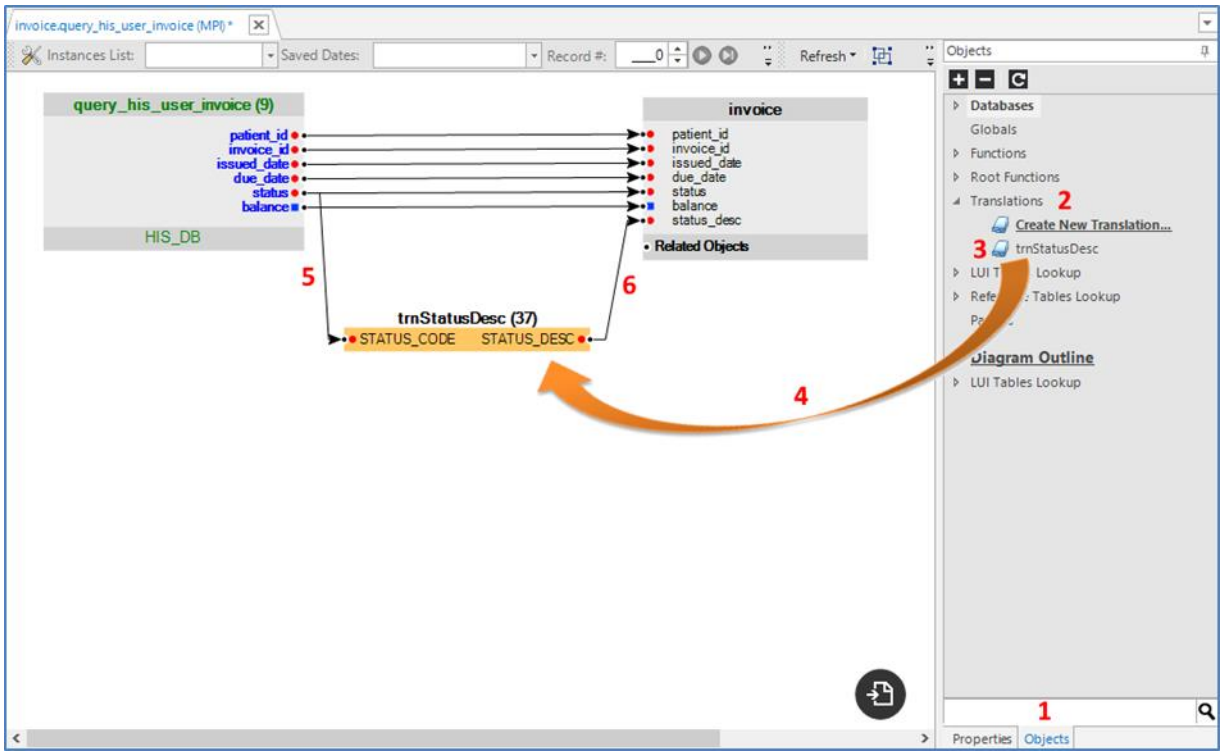
(2) Click the **Translations** item.

(3) Select the required translation from the translations list.

(4) Drag the translation into the working area.

(5) Connect input links from the source columns to the translation input parameters.

(6) Connect output links from the translation output parameters to the LU columns.



☞ To add a new Translation to the table mapping

There are 2 ways to add a new translation from the table population window:

(1) Right click anywhere in the Map. A pop-up menu appears. Then, select **Insert New Translation.**

(2) Click the **Create New Translation** option under Functions in the object tab. A new translation window is opened.

(3) Define the translation as described in Section 9.3 and click '**Save and Close**'. The translation is added automatically to the table mapping window.

(4) Continue with step 6 of Section 8.7.4.

8.7.5 Adding Global Parameters

You can add any global defined under the specific LU or under shared objects.

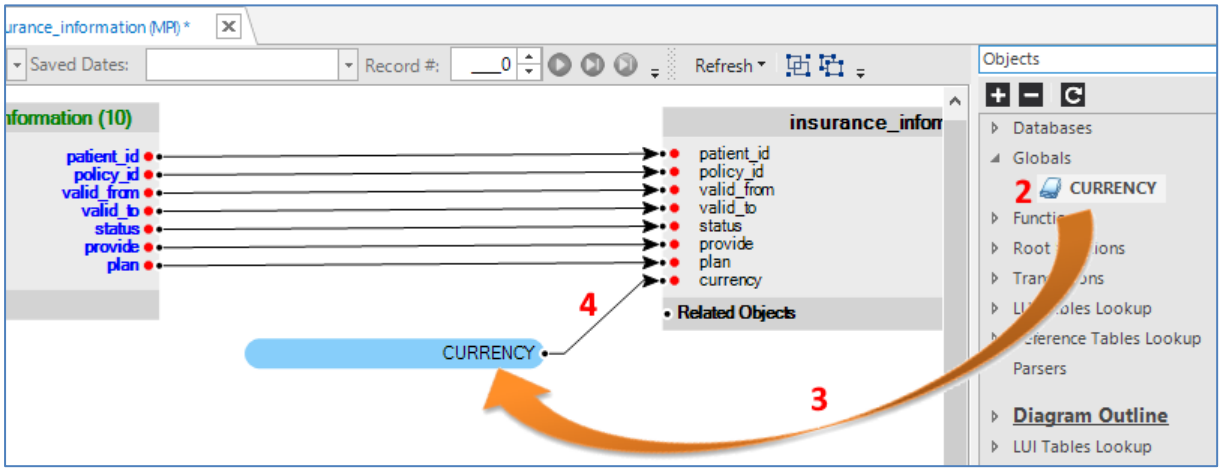
☞ To add a Global transformation to the table mapping

(1) Select the **Objects** tab in the right panel of the mapping window.

(2) Click **Globals** and select the required global from the **Globals** list.

(3) Drag the global into the working area.

(4) Connect the output link from the global to the table column.



8.7.6 Adding Constant

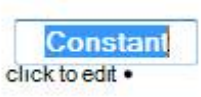
☞ **To add a constant into a table mapping**

(1) Right-click anywhere in the Map.

(2) A pop-up menu appears.

Select ***Insert Constant***

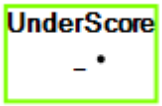
(3) The following text box appears in the workspace:



(4) Change the "**Constant**" to a name of the constant.

(5) Click the "**click to edi**t" text.

(6) Change the text to the constant's value.



(7) Connect an output link from the constant field to the input parameter or table column that will be set to the constant.

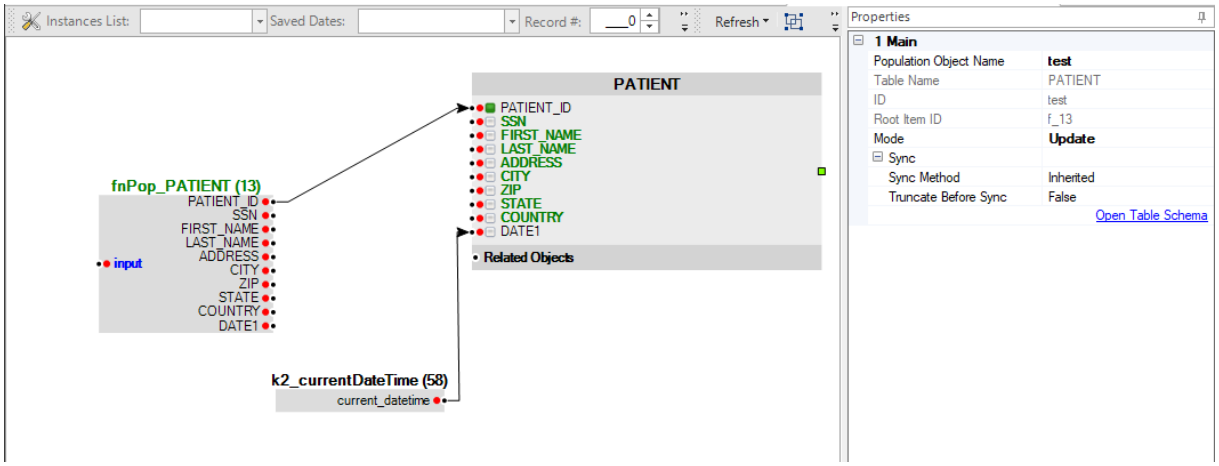
8.8 Working with Output Settings

All table properties and column properties can only be enhanced within the Table definition window, with the exception of the Mode property, which is defined for each Table Population:

- **Insert –** this is the default mode. The extracted record is inserted as a new record into the table.
- **Upsert –** the system checks via a key field if the record already exists in the table. If the record does not exist, then the record is inserted into the table. If the record already exists, then the existing record is updated.
- **Delete -** an entry in a database table.
- **Update** - an entry in a database table.

(1) Change the mode attribute to update/delete/upsert.

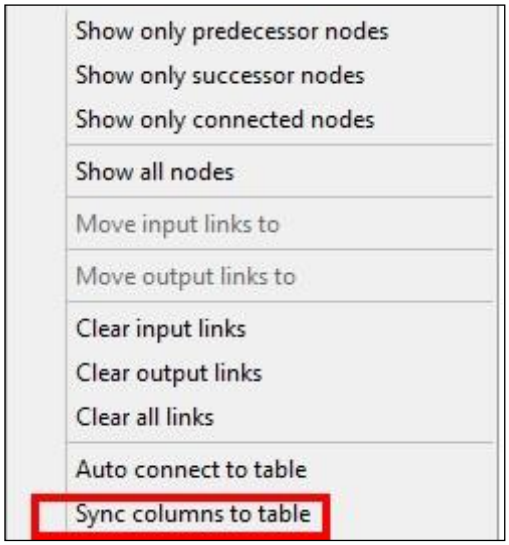
(2) Drag the lines that define the condition and select those fields in the target table.



8.8.1 Adding Columns to the Target Table

There are two ways to add columns to the target object:

- Right click the input object and click the Sync columns to table. This action creates the input table's columns in the table object, and links input table columns into table object columns.



> **Note:** This method automatically enables you to add a new field to the table Schema through the Table Population object. You are prompted to approve that the previous table setting can be overridden.

- Manual adding and enhancing of columns can only be performed within the Table definition window.

8.9 Saving the Table Object

Then click the **Save** icon or press CTRL+ S.

Then you need to populate the Name in the New Item popup window.

> **Note:** When you change a file name **after** saving the table using the "Save as" option, a new file is created with the updated name.

8.10 Debugging

As with any development environment, providing a tool that allows you to verify the effectiveness of the definitions is an important element. The K2View Fabric Debugger allows you to review the flow of data in the Table Population object and to insert various test cases. The debugger displays the data retrieved from the source and mapped into the table for a single instance.

Debugging is available only for tables attached to an LU.

☞ To debug the mapping of data for an Instance

(1) Click the **Enable Debug** icon ( ) in the second row of the Table Population window header.

(2) Select an Instance value from the **Instances List**, or enter a new instance value.

> **Note:** The **Instances List** is a drop-down menu that lists any instances that have previously generated data files for different instances. You can enter a new instance value to check the mapping for a different instance not listed.

(3) Select a previously generated file for this instance from the **Saved Dates** list.

> **Note:** The **Saved Dates** is a drop-down menu that lists the previously generated files (named by their timestamp) that may be used to debug from. Using an existing file is quicker as the data is extracted from the source system only for the debugged table.

(4) Click the **Play** button (at the right end).

> **Note:** There are three Play buttons with the following characteristics:

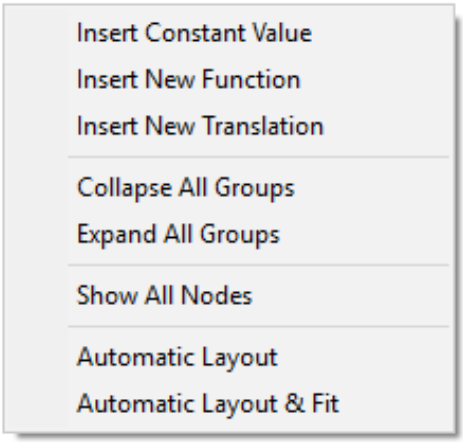
- Play - results for the first line are displayed in the data in the table window.
- Play - to debug and view the result of the next line for the same instance.
- Play - to debug and view the result of the same line for the next instance.

8.11 Viewing the Data Map

☞ To reorganize the mapping display to make it easier to view

(1) Right-click anywhere in the workspace of the **Table Population** window.

The Context menu is displayed.



(2) Click any of the following options to change the display:

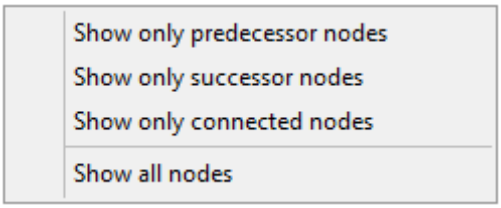
Collapse All Groups	Collapses groups.
Expand All Groups	Expands groups, to display members.
Automatic Layout	Arranges table items in an orderly manner.
Automatic Layout and Fit	Arranges table items in an orderly manner and adjusts Table Population resolution to fit the screen.

8.12 Filtering Mapping Rules

☞ To limit the number of objects displayed in the window

(1) Right click any object or field in the Table Population window.

The Context menu is displayed.



(2) Select one of the following display options:

Show only predecessor nodes	Show only objects and links, which are predecessor to this object\field(s).
Show only successor nodes	Show only objects and links, which are successors of this object\field(s).
Show only connected nodes	Show all objects and links connected to this object\field(s).
Show all nodes	Show all objects and links of the table.

8.13 Grouping Translations and Functions

In some cases, a table object may have many Translations/Functions. To make the table clearer and manage different combinations, the user can group and ungroup Translations/Functions.

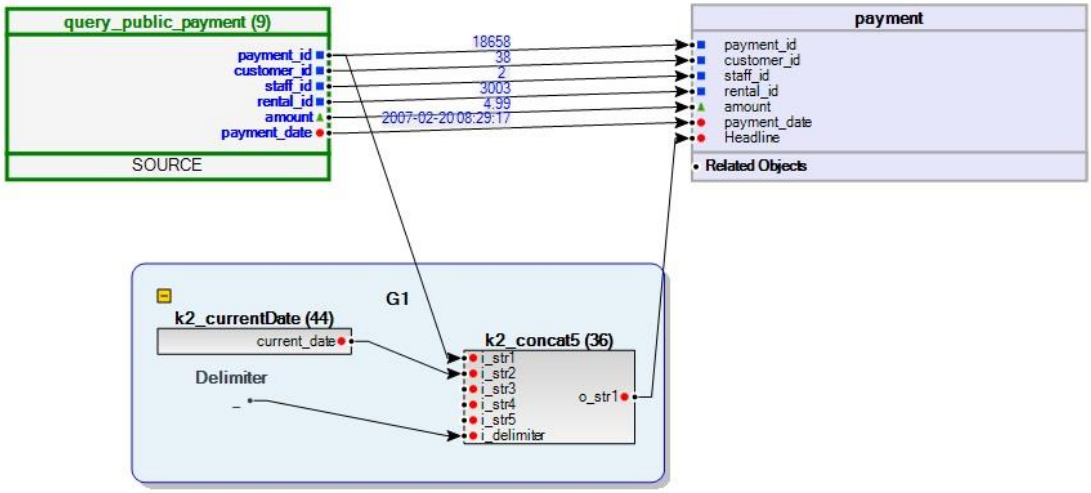
☞ To group a number of Translations/Functions into a single Group object

(1) Select the objects in the table that need to be joined into the group.

(2) Click the Group button, at the right end of the Toolset row of the window header.

(3) Double-click GroupedSubGraph.

(4) Edit the name of the Group object.



(5) Click the **Save** icon or press CTRL+S.

8.14 Ungrouping Translations and Functions

☞ To ungroup a Translation/Function group

(1) Select the objects in the Map to ungroup.

(2) Click the **Ungroup** button, at the right end of the Toolset row in the window header.

The Translations/Functions are not displayed as a group.

(3) Click the **Save** icon or press CTRL +S to save the update.

8.15 Automatically Connecting Fields

By default, when a new table is added to the Schema, links are automatically established between the columns of the Source DB and the similarly named columns in the LU table. During the stages of editing the links, for example, when you add different transformations via functions and/or translations, some of the links may be deleted. You can automatically reestablish the links between the identically named columns in the source and LU tables.

☞ To automatically connect identically named columns from source to a table

(1) Right click the **Source** table header.

(2) Select the **Auto connect to table** option.

(3) Click the **Save** icon or press CTRL +S to save the updated links.

8.16 Adding Links to a Table

Some tables may have a large number of columns in either the source or target. If all of the mapping lines are displayed it may become cluttered. The Studio supports a method to establish a link between the source and target without drawing the mapping line.

☞ To add output links from a table, without drawing the lines

(1) Right click a column in the source table.

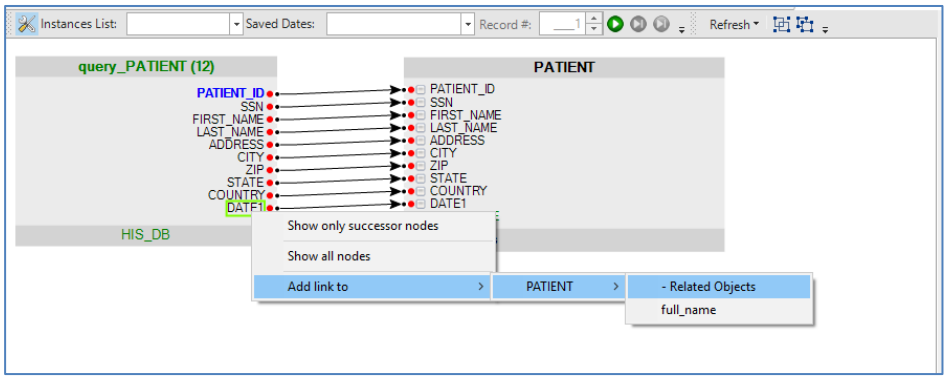
The Context menu is displayed.

(2) Select the **Add Link To** option.

A Menu with all tables that have unconnected columns is displayed.

(3) Select the table that includes the target column.

A Menu listing all unconnected columns is displayed.



(4) Select the target column.

☞ To add input links to a table, without drawing the lines

(1) Right click a column in the target table.

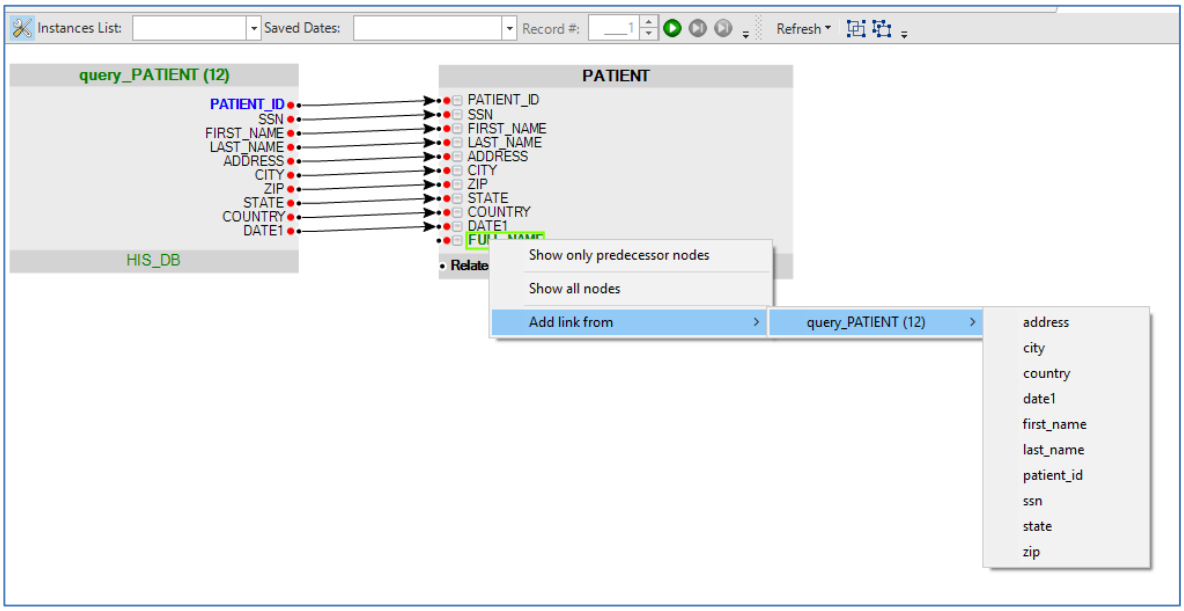
The Context menu is displayed.

(2) Select the **Add Link From** option.

A Menu with all source tables that have unconnected columns is displayed.

(3) Select the table that includes the source column.

A Menu listing all unconnected columns is displayed.



(4) Select the source column.

8.17 Moving Input / Output Links to an Object

This feature enables users to move input and output links from one Table Population object to another.

☞ To move an Input / Output link to another object

(1) Right click a column in the source table.

The Context menu is displayed.

(2) Select the **Move input links to** (or **Move output links to**) option.

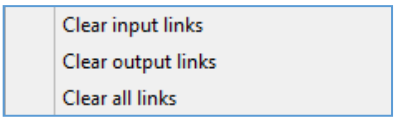
A List of possible target entities is displayed.

(3) Select the required target.

> **Note:** K2View Fabric Studio attaches an ID to an object (see example above). Therefore, each object instance has a unique identifier, which allows the same Function, Translation, or Lookup Object to be dragged into the map more than once.

8.18 Clearing Input/ Output Links

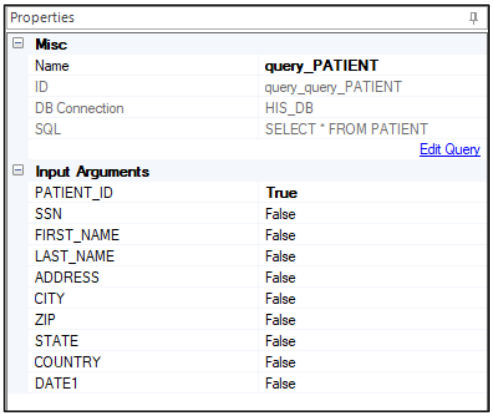
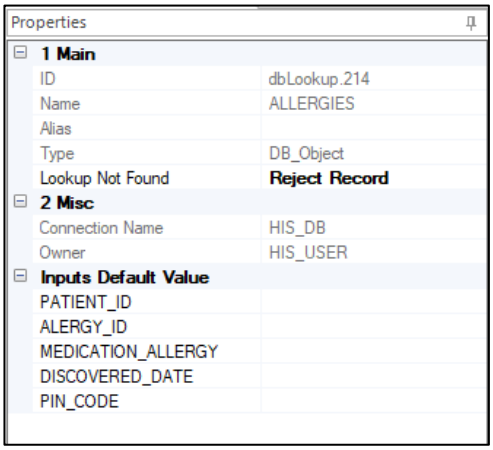
This feature enables users to clear input and output links from any map object.



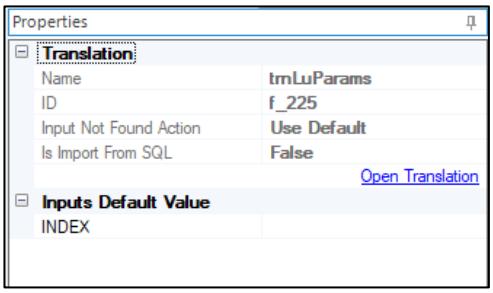
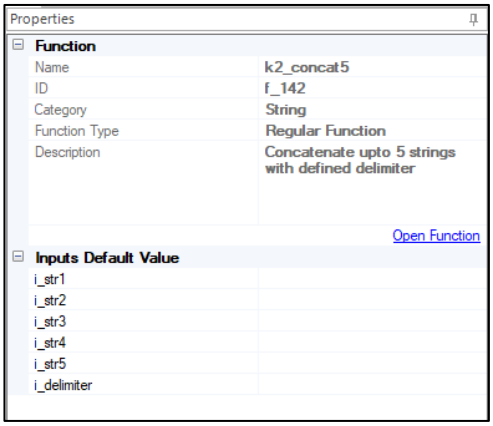
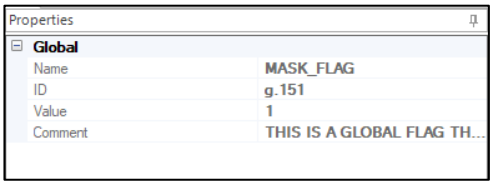
Clear input links	Clears all input links from Objects.
Clear output links	Clears all output links from Objects.
Clear all links	Clears all links (both input and output) from Objects.

8.19 Table Properties

When selecting an object in the Table Population object, the object properties become available in the upper right section of the table window. Each object has different properties, some may be editable, and some are grayed and displayed for information only. The following outlines different object types and their properties:

Object Name	Properties screen	Properties Description
Input DB Query		1. **Name-** name of the DB query object. Can be edited by user. 2. **ID-** internal ID for the object. 3. **DB Connection-** DB name for input table. 4. **SQL-** SQL query which retrieves data from the input table. User can edit the SQL query by clicking Edit Query. 5. **Input Arguments-** an input argument must be set to **True** to enable a link of this argument (field) to other objects in the Logical Unit (LU DB). If an input argument is set to **False**- user cannot link it in the LU DB. **For a root table in a LU DB- only one field can be set to 'True' in the input DB query of the root table.**
DB Object Lookup		1. **Main**- > **Internal ID**, name and type of the Lookup object > **Lookup Not Found-** defines the required treatment if no record is returned by the Lookup. The following options are available- > **Reject_Record**- this is the default option. The record is rejected and is not inserted into the target table. > **Reject_Instance**- the entire Logical Unit instance is rejected and is not loaded into the K2View Fabric Data Base. This option is used for critical errors. For example- users can define that if no billing address is found for a customer, then the customer data instance must be rejected.

Object Name	Properties screen	Properties Description
		> **Continue**- ignore error and load record. > **Report_N_Continue**- report a message to the user and continue with the load. User can check the issue later based on the reported message. 2. **Misc-** source DB details 3. **Input Default Value-** users can define a default value for each input field of the Lookup, which is used in the event there is no link connected to the field, or in case a null value is being passed. The value can be set to null or to an empty string by right clicking the field and choosing the required option. **Note:** When deleting the value set for this field using **Backspace**, the default is set to an empty string and is NOT set to null.
LUI Lookup		Properties are identical to DB object Lookup except for the following: 1. **Properties list** does not contain source DB details. 2. **Open Lookup Object**- enables the user to open the table object of the lookup into a separate tab.

Object Name	Properties screen	Properties Description
Translation		1. **Translation-** main attributes of the transaction object. 2. **Open Translation-** user can open the translation object into a separate tab. 3. **Inputs Default Values-** Users can define a default value for each input field of the Translation, which is used in the event there is no link connected to the field, or in case a null value is being passed. The value can be set to null or to an empty string by right clicking the field and choosing the required option. **Note:** When deleting the value set for this field using **Backspace**, the default is set to an empty string and is NOT set to null
Function		1. **Function** - Displays different properties of the function and their codes. 2. **Open Function** - Opens the function editor for this function. 3. **Inputs Default Values-** Users can define a default value for each input field of the Function, which is used in the event there is no link connected to the field, or in case a null value is being passed. The value can be set to null or to an empty string by right clicking the field and choosing the required option. 4. **Note:** When deleting the value set for this field using **Backspace**, the default is set to an empty string and is NOT set to null.
Global		Displays different properties of the global and its value.

8.20 Using Notes and Balloon Callouts

To add a note: Click  and enter the desired text.

To add a balloon: Click  and enter the desired text.

9 Working with Transformations

9.1 Overview

Functions and Translations enable you to perform transformation rules from the source systems into K2View Fabric. For example, you can define transformations that perform complex calculation, fetch data from the source systems or the in-memory DB, update tables, manipulate strings, etc.

All functions are written in the Java language.

Functions can be defined as either:

- **Shared Objects** - A function that is defined under the Shared Objects can be utilized within all Logical Units under the specific project.
- **Logical Unit** - A function that is defined under a Logical Unit can be utilized only within a Logical Unit.

When a function or translation is defined both in the **Shared Objects** and under the **Logical Unit** with the same name and parameters, the **Logical Unit** function code is executed.

9.2 Working with Functions

9.2.1 K2View Fabric Studio Functions

K2View Fabric Studio supports three classes of functions:

- **Built-In Functions** – Detailed information for these functions, including parameters and descriptions, is provided in Chapter 22.
- **Product Functions** – Detailed information for these functions, including parameters and descriptions, is provided in Chapter 21.4.2.
- **Project Functions**: You can add customized functions to perform complex activities within a table object or to select queries on a specific instance ID. You can add as many new customized functions as necessary and export them to a different project if required. These functions are also visible in the tables or in the Java editor window.

  A Function may have different scopes. The function may be defined as a Shared Object that can be used in any LU of the project. If the function is defined within a particular LU then it is accessible only within that particular LU but not within other LUs.

9.2.2 Creating Functions

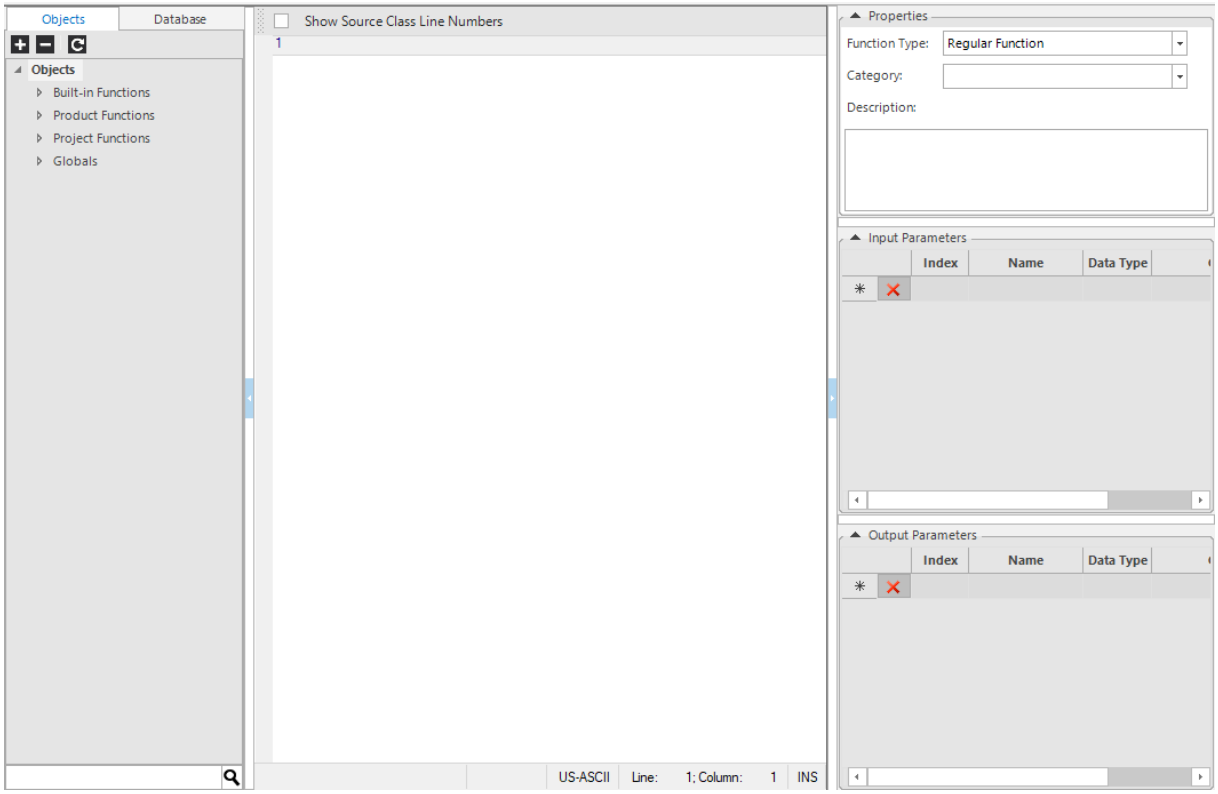
To create a new project LU function

(1) Right Click the Functions in the project tree, and then select the **New Function** option in the context menu.

The Context menu is displayed.

(2) Right click the Functions option under each LU in the project tree.

***The Function Manager** window is displayed.*



(3) Select **Function Type** for the function from the dropdown menu. The following options are supported:

- Regular Function
- Root Function- see more details in section 9.2.6
- LUDB function- register a function as a DB function
- Decision Function- user function that the user can decide by java code if sync should occur using built-in functions –getMigrationID, getLastSyncTime, getTableName, getPopulationName. The sync is performed based on this code. When Decision Function is selected, the user needs to choose a function from a drop-down list of the available decision functions.

- Switch Function- you can create multi record types using a new function type "switch function" (enabled only in parser map). This option allows you to define deferent record types for the same parser.
- User Job – added in release 4.1 to support running a light weight function using the jobs mechanism.

(4) Select a category for the function (or enter a new category) from the **Category** drop-down menu.

(5) Enter the function description in the **Description** field (at the top of the window).

> **Note:** The description is not mandatory, but it is highly recommended to provide this information.

(6) Define the function parameters, as described in Section 0.

(7) Generate code for the function, as described in Section 9.2.3.

(8) Right click the working area, and then select **Validate Syntax** in the context menu to verify the validity of the function code syntax.

(9) Click the **Save** icon or press CTRL+S to save the function definition to the project.

A popup window is displayed: New Item. Populate the name of the function in the Name field.

> **Note:** For best practices, prefix the function name with “fn”. The name should not include any special characters (for example, ‘#’ or ‘$’).
>
> **Note:** DB functions may be utilized in the Logical Unit as an SQL function, thereby expanding the SQL library. For example- fnConcat function concatenates 2 fields and is registered as a DB function. The user can define the following select when creating fnFunc2: “select field1, field2, fnConcat(field3, field4) from TABLE_ADDRESS”. The LU instance function must return a value.

☞ To create a new project Shared Object function

(1) Right-click the **Functions** entry under **Shared Objects** in the Project Tree pane (the left side of the window).

The Context menu is displayed.

(2) Select the **New Function** option.

***The Function Manager** window is displayed.*

(3) Continue with Step (3) of the previous procedure.

9.2.2.1 Manually Defining Parameters

The Function Manager window includes the following two panes on the right side of the window:

- **Input Parameter** pane.
- **Output Parameter** pane.

Both panes include the following components:

- **List panel** – displays a list of parameters that have been defined. Each parameter is listed by name, data type, and comment.
- Entry fields – Text entry fields for a parameter name, data type, and comment.

To manually define parameters for the function

> **Note:** The following procedure describing the steps to Add Parameters applies to both input and output parameters, using the Input Parameter pane or the Output Parameter pane, respectively.

(1) Enter the parameter name in the **Name** text entry field.

> **Recommendation:** Best practice is to prefix names of input parameters with "i_" and output parameter names with "o_".

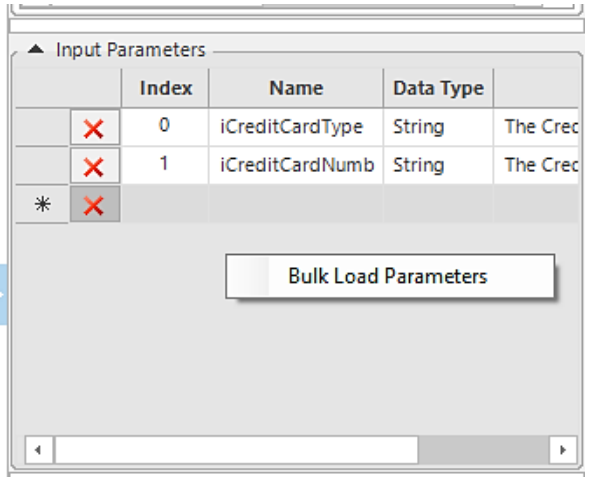
(2) Select the data type for the parameter in the **Data type** field.

(3) Enter a comment describing the parameter purpose in the **Comment** field.

(4) Click the **Save** icon or press CTRL+S to add the parameter to the parameter list.

> **Note:** To add additional parameters repeat Steps 1-4 of this section.

To manually bulk-load parameters for the function

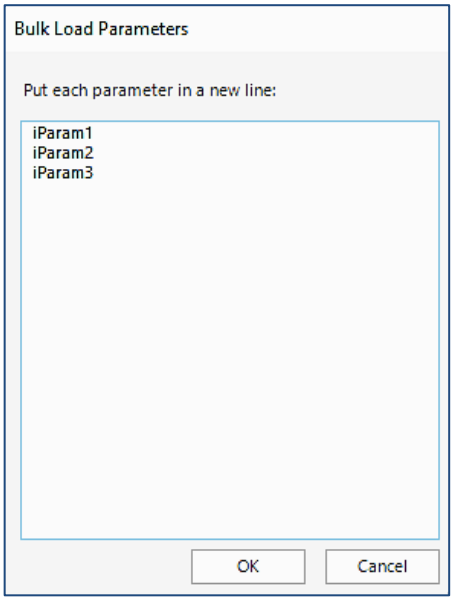


(1) Right-click the input/output parameters pane and click the **Bulk Load Parameters** button.

A popup window with a text area is displayed.

(2) Enter/paste a list of parameters and click **OK**.

The new parameters list is loaded to the parameters' pane with a string type.



(3) Click the **Save** icon or press CTRL+S to save changes.

> **Note:** This option allows the user to write/paste argument names, one per line, where all arguments are created of type String. Each parameter can then be updated separately.

9.2.2.2 Automatic Parameter Definition

The Objects/Database tree pane is displayed on the left side of the Function Manager window. This window displays the hierarchy of objects defined in the LU or tables in the source DB, based on the selection of two tabs at the top of the pane.

☞ **To automatically add a function parameter based on Database column**

(1) Click the **Database** tab in the Objects/Database pane (on the left side of the window).

(2) Select Interface from the **DB Connection** drop-down menu (at the top of the pane).

A Hierarchy of database tables is displayed.

> **Note:** You can also select Newest LUDB File from your DB connections dropdown. In order to use the Newest LUDB File, you must run Data Viewer at least once.

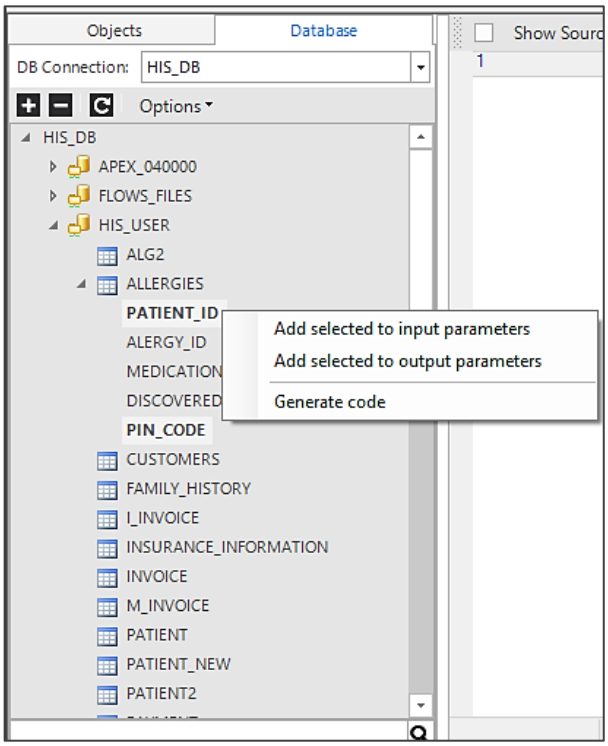
(3) Click a table in the database hierarchy.

(4) Click the name of a column to use as a basis for the parameter.

> **Note:** Multiple columns can be selected by pressing the Ctrl-Shift key combination.

(5) Right-click the selected column name.

The Context menu is displayed.



(6) Select **Add selected to input parameters** (or **Add selected to output parameters**) to add to input (output) parameters list.

New parameters added to parameter list in relevant parameter pane. The new parameter name is the name of the column and data type is the column data type.

9.2.2.3 Updating Parameter Definitions

☞ **To update the parameter definitions**

(1) Select a parameter in the parameter list.

Parameter name, type, and comment are displayed in the text entry fields.

(2) Edit the text entry fields.

(3) Click the **Save** icon or press CTRL+S to save your update.

A Changed parameter definition is displayed in the parameter list in place of previous values.

> **Note:** Updates to the input/output parameter name are automatically applied to all occurrences of the parameter in the function java code.

9.2.2.4 Deleting Parameters

☞ **To remove a parameter from the parameter list**

(1) Select a parameter in the parameters list.

Parameter name, type, and comment are displayed in the text entry fields.

(2) Press the **Delete keyboard** key or click the  icon next to the parameter.

A Parameter is removed from the parameters list.

9.2.3 Editing a New Project Function

9.2.3.1 Manually Creating a Project Function

The main working area of the Function Manager window acts as a Java editor for the programmatic logic of the function. Java code may be entered into this working area. For more information on the Java editor, see Section 17.1.

In addition, developers may automatically generate the base Java code and then edit this code to customize it for the specific functionality.

9.2.3.2 Automatically Generated Code

☞ **To automatically generate code for the function**

(1) Click the **Database** tab in the Objects/Database pane (on the left of the window).

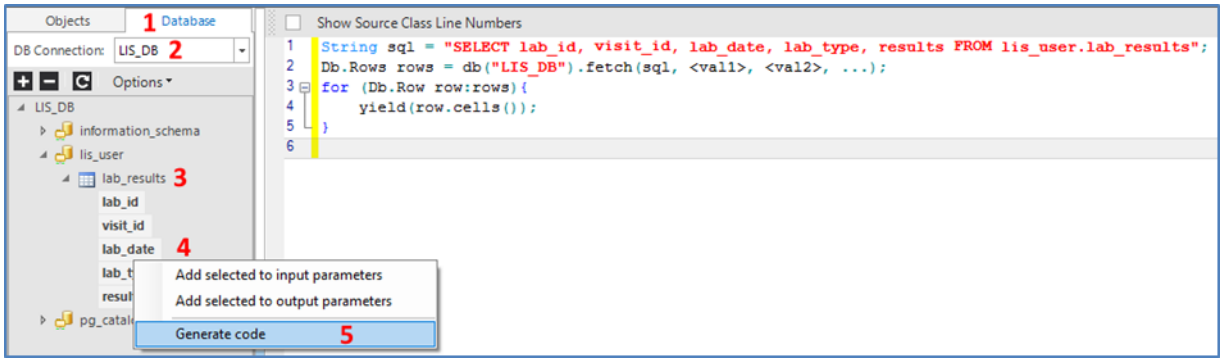
(2) Select Interface from the **DB Connection** drop-down menu (at the top of the pane).

(3) Click a table in the database hierarchy.

(4) Click a name of the column to use as a basis for the parameter.

> **Note:** Multiple columns can be selected by pressing the Ctrl-Shift key combination.

(5) Right-click the selected column name.



(6) Select the **Generate code** option.

Generates code for a Select query and a For loop that is displayed in the main working area of the window.

See the example below for a generated code:

Select PATIENT_ID and PIN_CODE of ALLERGIES table of HIS_DB DB interface and generate code.

The following code is generated:

String sql = "SELECT PATIENT_ID, PIN_CODE FROM ALLERGIES";

Db.Rows rows = db("HIS_DB").fetch(sql, <val1>, <val2>, ...);

for (Db.Row row:rows){

}

You need to populate your parameters for the query in <val1>, <val2> , etc. If you do not need any parameter for your query- you need to remove the <val1> and <val2> place holders.

> **Notes:**
>
> > **Where** clause should be added manually to the generated query within the java code.
>
> > **<val1>, <val2>,…** are place holders for parameters for your query. If you do not need any parameter for your query- you need to remove the <val1> and <val2> place holders from your code and send the fetch command only the 'sql' parameter.

9.2.4 Updating Functions

Before updating any function, keep in mind that any update to a function can impact maps, or any other object, where the function is used. If the change is in the body of the function, verify that the updated logic is applicable to all other objects that the function may be applied to. If the update is to add or remove parameters, then other function calls will be updated.

Function Name cannot be updated after the function is saved. Changing a name for an existing function after it has been previously saved, creates a new function with the updated name.

To access the function in the Function Manager window:

- Double-click the function in the project tree

  or

- Double-click the function from the map.

9.2.5 Deleting a Function

☞ **To delete a function from the project**

(1) Right click the function in the project tree.

(2) Select **Delete Selected Items.**

> **Note:** If the function is referenced within other objects (Table or Function), a pop-up message appears prompting you to delete the references as well.

9.2.6 Understanding Root Functions

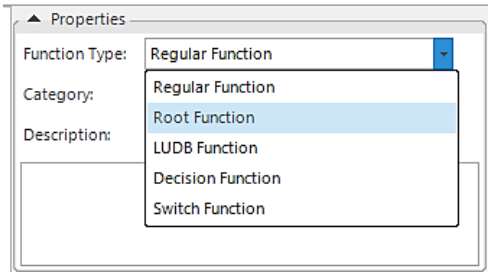
A root function can be used as an input for the K2View Fabric table. Each table object can be populated by a source table or by a root function. A root function can perform a *Select* on a source table and/or perform additional manipulations and calculations. Each record returned from the function is inserted into the table.

9.2.6.1 When to Use a Root Function Rather than a DB Query?

- When the input data is being extracted from a file or any other external source (Not DB).
- When complex aggregations are required in order to determine the number of records which should be loaded into the K2View Fabric table.

- You can set a function as a root function by selecting the option root function from the dropdown list under Function Type.

  A root function must have at least one input and one output parameter and must yield an object.



  The root function main **select** and **loop** can automatically be generated as mentioned above.

  An example for a generated code of a root function which returns PATIENT_ID, SSN, and FIRST_NAME from PATIENT table:

```
String sql = "SELECT PATIENT_ID, SSN, FIRST_NAME FROM PATIENT";
Db.Rows rows = db("HIS_DB").fetch(sql);
for (Db.Row row:rows){
    yield(row.cells());
}
;
```

9.2.7 Understanding Enrichment Functions

Enrichment functions are functions without input/output parameters that are used to insert, update, or delete from a table an object that has already been loaded. For example, a function that calculates the total amount of payments for a customer and updates the field of total payment amount on a customer table.

- The enrichment functions are executed **after** the population of the tables in the Logical Unit, and, therefore, can extract data from any table within the Logical Unit. The execution order of those functions is determined at the Schema level as described in Section 7.6.
- The user can attach the enrichment function to a table in the enrichment drop down list in the table property (see Section 8.2.4).

> **Note:** In case the function updates one or more fields in the table, it is recommended to mark those fields as **ComputedField** (In the field's properties window).

9.3 Working with Translations

9.3.1 Translations: Overview

Translations enable users to store decision tables that reside within K2View Fabric. All changes to this object are tracked under the K2View Fabric Studio Configuration Control.

In addition, because the translations are consolidated within K2View Fabric, they are in line with the concept of "All-In-One" – centralizing all project information in one location. This significantly improves translation management and oversight.

Project Translations may be defined either at the Shared Objects level or at the Logical Unit level. A Translation defined at the Shared Objects level may be used in any of the LU's included in the Project.

For the initial population of a Translation, the default option is to retrieve the translation information from a file. However, the translation information can also be retrieved from a database. This section describes how to create a Translation file using both of these methods.

The Translation definition is a two-part definition:

- Translation Schema – defines the translation parameters and indicates, using the schema properties (see Section 9.3), whether the parameter is an input or an output and what value types are supported.
- Translation Data – provides the actual decision table of how to translate different input combinations into the output values.

9.3.2 Defining a New Translation

☞ To define a new translation

(1) Right click the Translations option in the project tree and then select the **New Translation** option in the context menu.

(2) Define the parameters of the translation completing the Schema properties (Section 9.3.4) for each parameter.

(3) Click the **Translation Data** tab (on the left side of the window).

(4) Enter the input and output value combinations for all of the parameters.

(5) Select the action for values that are not found in the translation data in the drop-down menu at the bottom of the window.

(6) Click the **Save** icon or press CTRL+S to save the function definition to the project.

A popup window is displayed: New Item. Populate the name of the translation in the Name field.

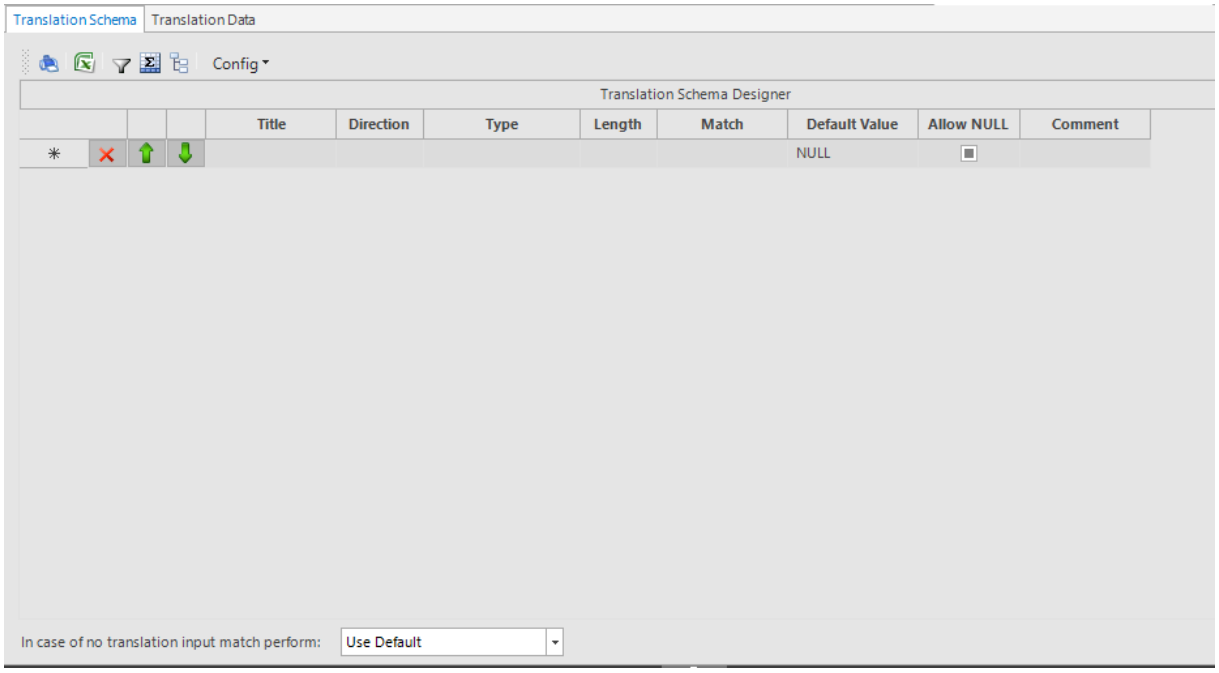
> **Note:** The name must not include any special characters (for example, # or $), and it is recommended that the name starts with “trn”, for example, “trnMarket

9.3.3 Translation Window

The Translation Window is used to define the Translation and save it to the K2View Fabric Studio Project Tree.

The Translation window includes the following two tabs along the top of the window to select between:

- **Translation Schema** – a window to define parameters for translation.
- **Translation Data** – a window to enter data for input and output combinations.



At the top of the pane the following icons are included:

Icon	Action
	Print definition.
	Export definition as an Excel file.
	Filter definition by one or more columns.
	Toggle groupings.
	Toggle summaries.

At the bottom of the window, the following element is displayed:

- Drop-down menu to define what happens when no translation is found. The following options are available:

Option	Purpose
Use_Default	Use default value.
Reject_Record	Rejects a single record.
Reject_Instance	Rejects the entire instance.
Report and Use Default	Reports the missing value and uses the default value.

9.3.4 Translation Schema Properties

Each Translation parameter is characterized by the following set of parameters in the Translation Schema window:

Parameter Property	Description
Title	Name of the parameter
Direction	Indicates whether parameter is an Input or Output.
Type	Value type. The following types are supported: ■ Integer ■ Real ■ Text ■ Blob (for DB that support the type) ■ Auto increment
Length	Field length - max number of characters
Match	A regular expression that indicates a restricted format for the values

Parameter Property	Description
Default Value	Default value of the field. This value is used if no entry is found (only relevant for the Output parameter).
Allow NULL	True/False indicates if parameter can be NULL. The value is set to allow NULL by default.
Comment	Any text comment for explanation

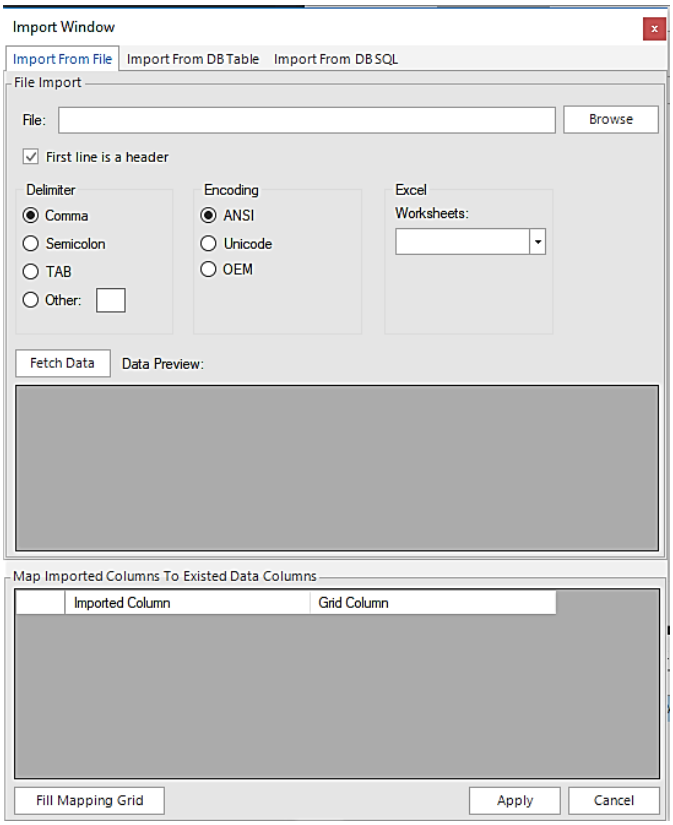
The following icons appear next to each translation parameter:

Icon	Action
	Deletes the parameter.
	Moves the translation parameter upwards.
	Moves the translation parameter downwards.

9.3.5 Importing Translation Data from a File

(1) After you complete the definition of the translation schema, open the Translation Data tab and click the 'Import Data' button.

(2) Select the Import from the File tab at the top of the window.



(3) In the main area:

(b) Enter the location of the file, or click Browse to look for the file.

(c) Mark the first line as a header check box if the file consists of a header that will be skipped when importing the data.

(d) In the delimiter list, select the file delimiter.

(e) In the Encoding list, select the file's encoding.

(f) If the file is an Excel file, select the worksheet page from the drop-down list.

(g) Click Fetch Data to preview the imported file.

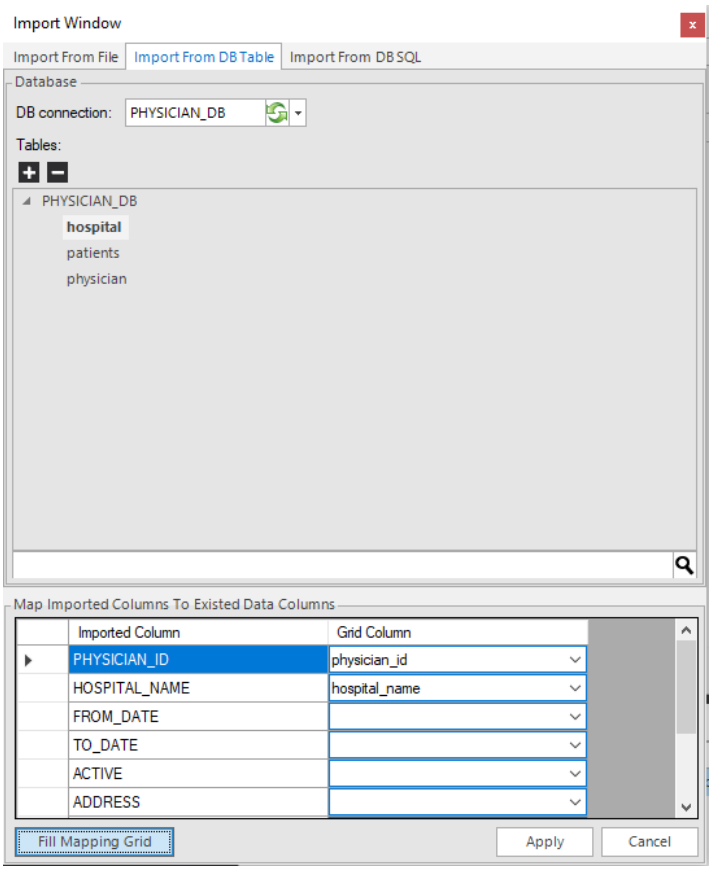
(4) In the mapping area:

(a) Click **Fill Mapping Grid** to create the association between the fetched data and the translation fields.

(b) Click **Apply** to commit the association between the fetched data and the translation fields. A confirmation window appears indicating the number of rows imported.

(c) Click the **Save** icon or press CTRL+S to save the translation.

9.3.6 Importing Translation Data from a Database Table



(1) Click **Import from DB Table** at the top of the window (1).

(2) In the **Database area** (2):

(a) Select the DB Connection from the drop-down list.

(b) From the Source tree, select the table that has the data to be fetched.

(3) In the mapping area (3):

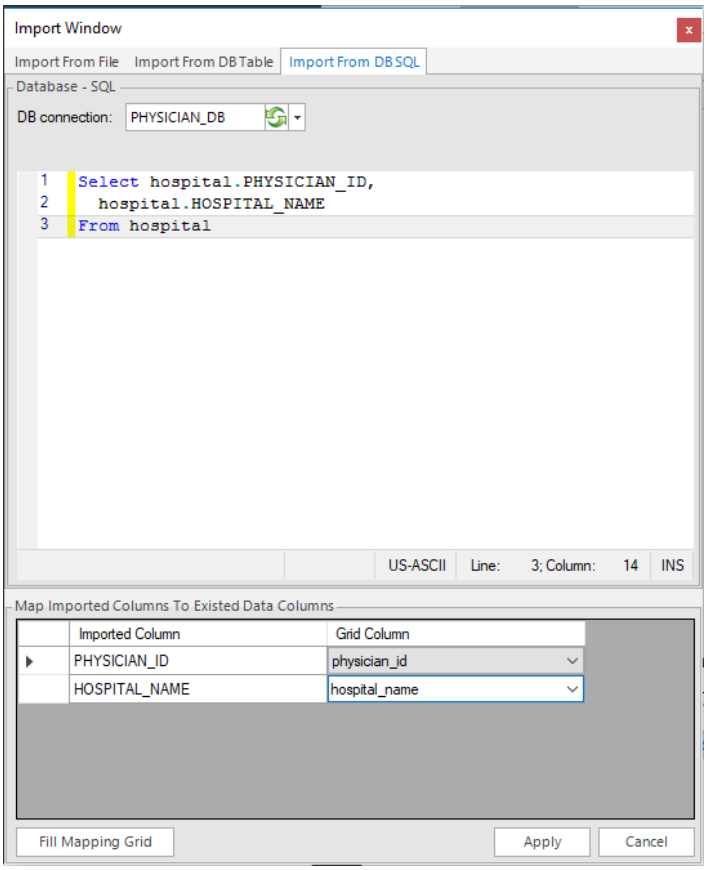
(a) Click **Fill Mapping Grid** to create the association between the fetched data and the translation fields.

(b) Click **Apply** to commit the association between the fetched data and the translation fields.

(c) Save your changes. ( )

9.3.7 Importing Translation Data Using DB SQL

☞ To import data from the database using an SQL statement



(1) Click **Import from DB SQL** at the top of the window (1).

(2) In the **Database** area:

- Select the **DB Connection** from the drop-down list,
- Enter the **SQL Query** to fetch the data.

(3) In the mapping area click **Fill Mapping Grid** to create the association between the fetched data and the translation fields.

(4) Click **Apply**.

(5) Save your changes. (0)

9.4 Working with Globals

9.4.1 Editing Globals

Globals are the pre-defined variables that can be defined at two levels of a scope –

- Shared Objects – available to all LUs throughout the project.
- Logical Unit – available within the specific Logical Unit.

  If a Global is defined as both the Shared Object and Logical Unit item, the Logical Unit definition is used within the scope of that Logical Unit. Other Logical Units use the Shared Object.

To create a new global

(1) Right click the Globals option in the project tree.

(2) Enter a name in the **Name** column.

(3) Enter a constant value in the **Value** column.

(4) (Optional) Enter a category in the **Category** column.

(5) (Optional) Enter a comment in the **Comment** column.

(6) Global can be marked as final or not. If it is not marked as final, it can be overridden during the execution and implementation using set_global global '<LUT NAME>.<PARAM_NAME>[=<PARAM_VALUE>]' command.

(7) Click the **Save** icon, or press CTRL+S to save the global to the project.

9.4.2 Use of Globals in Java Code

This section, describes the use of the Globals that are defined by the user within the functions or SQL query. The format is derived from the name of the Global.

Here is an example or some basic Globals in the following screen:

	Category	Final	Name	Value	Comment
▶ ✕	MASKING	☑	MASK_FLAG	1	THIS IS A GLOBAL FLAG THAT IS USED TO DETERMI
✕	SHARED MEMORY	☑	SHARED_MEMORY_HOST_NAME	localhost	The host name/ip where the shared memory service (me
✕	SHARED MEMORY	☑	SHARED_MEMORY_HOST_PORT	6379	The port of the shared memory service (memchace).
✕	SHARED MEMORY	☑	SHARED_MEMORY_COMPRESSED_FLAG	0	Boolean indicator used to determine if to compress the va
✕	SHARED MEMORY	☑	SHARED_MEMORY_EXPIRATION	86400	The expiration time of values in the shared memory.
✕	MASKING	☑	REPLACE_APOSTROPHE_FLAG	1	Determines if to convert a single apostrophe to double ap
✕	MASKING	☑	MAX_NUM_OF_ITERATION	10000	This global is defined in order to prevent infinite loops
✕	MASKING	☑	DB_INTERFACE_NAME	ludb	The interface of the DB on which DBExecute, DBQuery ...

This global is used within the function codes as follows:

```
String maskFlag = MASK_FLAG;
```

The variable `maskFlag` receives the value '`1`'.

10 Working with Parser

This feature allows you to parse web log files or other dynamic files and insert the data from the requested columns into the database.

The Fabric server monitors a hot folder and fetches files in real time according to their names (as defined by an expression).

K2View Studio enables you to create a parser that takes such input files, identifies data elements in them, manipulates the content, and loads the files' data elements into a table in the Fabric database.

From the release 4.1 Parser support as input interface also: HTTP, JMS and KAFKA.

11 Fabric Job Mechanism

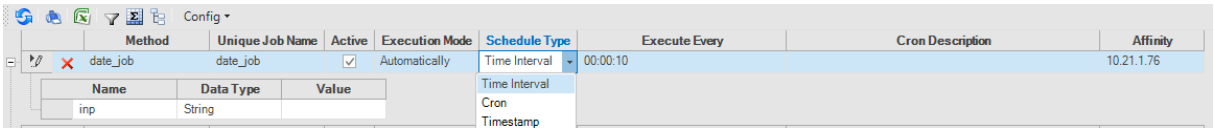
A new job mechanism was developed for Fabric. The job mechanism is currently used for the following:

- Parsers execution
- Process files from predefined directories using SFTP or Local File System interface types

A new job table- k2_jobs- is created under k2systems keyspace of Cassandra when Fabric starts.

12 User Job

- The ability to run light weight user functions without the need to execute parser.
- Function should be defined as 'User Job' function type to be included in the drop down list.
- Function can include input parameters.
- User jobs appears in Studio under LuType section in 'Jobs' section.
- Schedule Type
  - Time interval – job is scheduled to run every X time
  - Timestamp – job is scheduled to run one time on a given datetime
  - Cron – job is scheduled to run according to crontab command
- It is possible to define the AFFINITY for the User Job.
- User can decide if to run the Job manually or automatically after a deploy.
- User can decide if to mark 'User Job' as Active or not.



13 Parser Execution

13.1 Execution Mechanism

The Fabric parser mechanism creates a job for each parser execution. If the parser is set to run automatically, then a new record is created in k2_jobs table when the parser is deployed to the Fabric server. If the parser is set to run manually- then a new k2_jobs record is created for the parser when it is executed. Fabric allocates one available node to execute the parser.

Affinity – a new attribute is added to parser properties in the Fabric Studio- affinity. Fabric allocates an execution server based on the populated affinity. If the affinity includes several Fabric nodes- for example the DC name or the Logical node name contain several Fabric nodes- one of those nodes executes the parser. When affinity is not populated- Fabric distributes the parsers between different nodes of the Fabric cluster.

13.2 Configure the Parser Mechanism

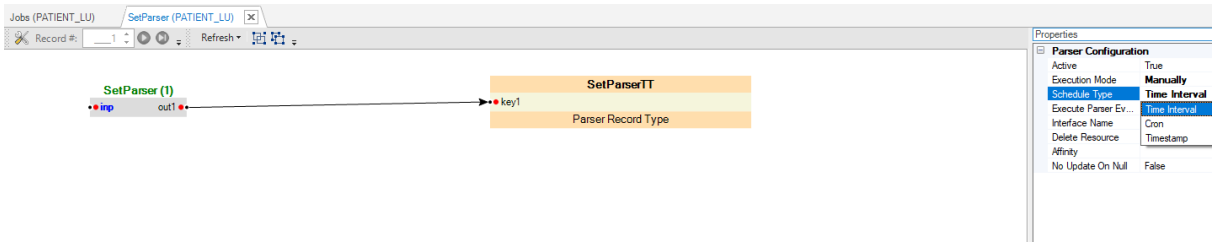
13.2.1 Adding a New File Parser

This section describes how to create a map that parses input files to produce a parsed file.

13.2.2 Defining the Parser Record Type

(1) Right-click the parser (below tables section) under any LU and click **New Parser Map**.

*The **Parser Map** tab displays.*



(2) Set parser properties:

- **Active**: Activates/deactivates the parser. Only activated parsers can be run.

(3) **Execution Mode**: Automatically or manually. When selected as manually, the parser should not run in case of deploy or fabric start. By default, it is set as automatically run. By default- execution mode is set to Automatically.

- Schedule Type
  - **Time interval –** parser is scheduled to run every X time.
  - **Timestamp –** parser is scheduled to run one time in a given datetime.
  - **Cron –** parser is scheduled to run according to crontab command.
- **Interface name:** add the name of the interface (SFTP or Local Directory interface type).
- **Delete resource**: delete the resource file after parsing the file.
- **Affinity:** can be populated by IP address, DC name, or logical name. Can have multiple values separated by a comma.
- **No update on null:** false by default. In case set to true, null fields will not be updated to avoid Cassandra tombstones.

(4) Edit parser record type: click the parser record type and edit the "Properties" fields in the right panel.

This record type defines the structure of the generated outcome file, after manipulating and parsing the original input files. Update the following attributes:

- **Name**: Holds the name of the parser record type. The name can be up to 48 characters in length, and is validated to ensure that no duplicate names are created.
- **Columns List**: Lists record column fields, including their properties.
- **PK Columns List**: Mandatory property that lists the columns in the parser map. The PK columns list can be used as input columns of a table populated, based on a parser.
- **Clustering Key Order**: Specifies whether clustering keys are sorted in the ascending or descending order.

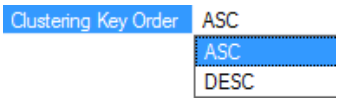
> **Note:** If the primary key is changed after a parser record is deployed, a message displays, prompting you to redeploy. In this case, you must drop the affected parser table and then redeploy all information into that table.

(5) Update the Columns List – Click the  icon in the columns list row.

The Columns List Editor is displayed. (A description for the columns list is provided below.)

(6) Specify additional primary keys in the **PK Columns List** field, as described on page 154.

(7) For clustering keys, specify whether to sort the keys in the ascending (**ASC**) or descending (**DESC**) order. For more details about clustering keys, see page 154.



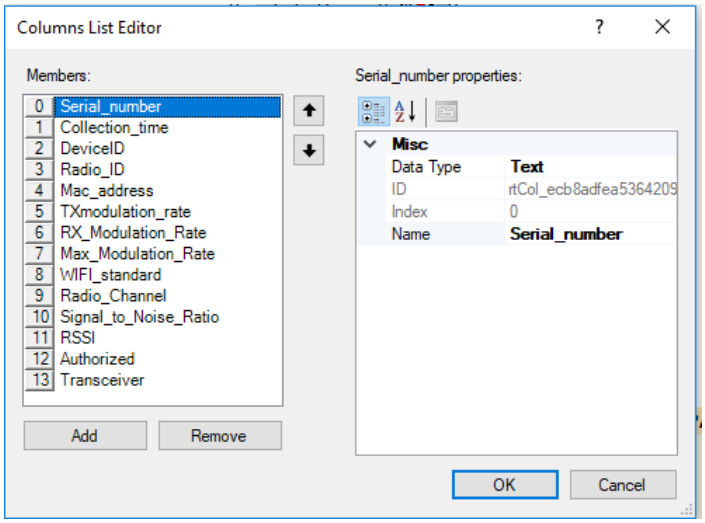
☞ To create/update the columns list

(1) Click **Add** to add a new member (column).

The column's properties pane displays.

(2) Edit the name and data type in the properties pane.

(3) Click **OK** to save the updates.



(0)

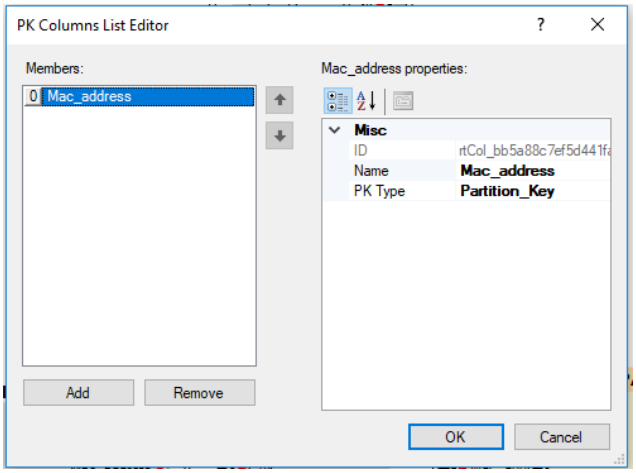
☞ To define primary key columns

You can define columns as a **primary key**, and designate whether it is a partition key or clustering key. More than one column can be added as a primary key. The first column that you specify in the **PK Columns List** is automatically added as a partition key. A primary key must contain at least one partition key.

By default, the **PK Columns List** is empty until you define one or more additional primary keys.

(1) Select the **PK Columns List** property and then click the Browse ... button.

(2) Click the **Add** button to add a new column. The Members pane specifies which columns are primary keys. When you click **Add**, a new column is added and is automatically assigned a default name, as shown below:



(3) In the Properties pane, you can assign a new name to the column. To do so, click the **Name** property and then select an existing column in the map.

*After making your selection, a unique ID is automatically assigned to the column and is shown in the **ID** field.*

(4) Specify whether the column is a partition or clustering key by clicking the **PK Type** property and then selecting the applicable key type in the dropdown menu.

13.2.3 Defining the Population of the Parser

K2View Studio provides three built-in functions that can be used for parsing an input file:

k2_FolderStreamReader – The function receives parameters for identifying and parsing the input files and produces each row of the file, separated into individual objects.

The function's input parameters are:

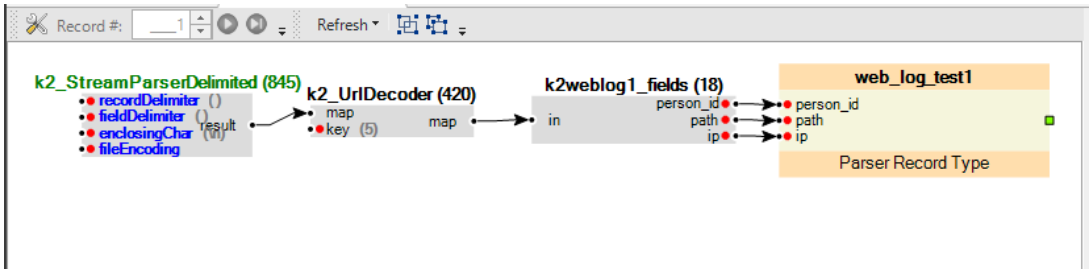
- **Folder path** – The path to the source folder to parse when executing the parser on the server.
- **FolderPathDebug** – The path to the source folder to parse for debugging purposes when using K2View Studio.
- **regexFilter** – A regular expression to identify the input files' naming pattern.
- **recordDelimiter** – Defines the delimiter string between each record in a file (for example, \n).
- **fieldDelimiter** – Defines the delimiter string between each field in a single record in the file (for example, " ").
- **enclosingChar** – Defines the character to be used as the enclosing value for the string.

k2_StreamParserDelimited – Input Stream function to call from the studio, close stream, and session after finish to read from stream.

The function's input parameters are:

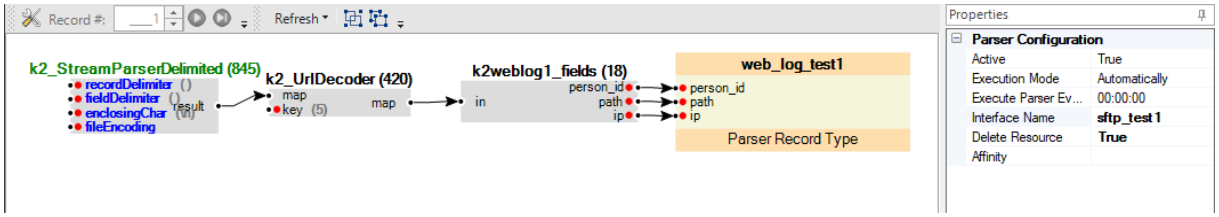
- **recordDelimiter** – Defines the delimiter string between each record in a file (for example, \n).
- **fieldDelimiter** – Defines the delimiter string between each field in a single record in the file (for example, " ").
- **enclosingChar** – Defines the character to be used as the enclosing value for the string.
- **k2_UrlDecoder** – Implemented specifically for delimiting URL web logs. This function receives an entry map, which was created based on a delimited file, as well as a key, which represents the URL position in the file. The function returns a map based on the URL keys and attributes (determined by delimiting = and &) on top of the original fields.

An example of a parser map is shown below:

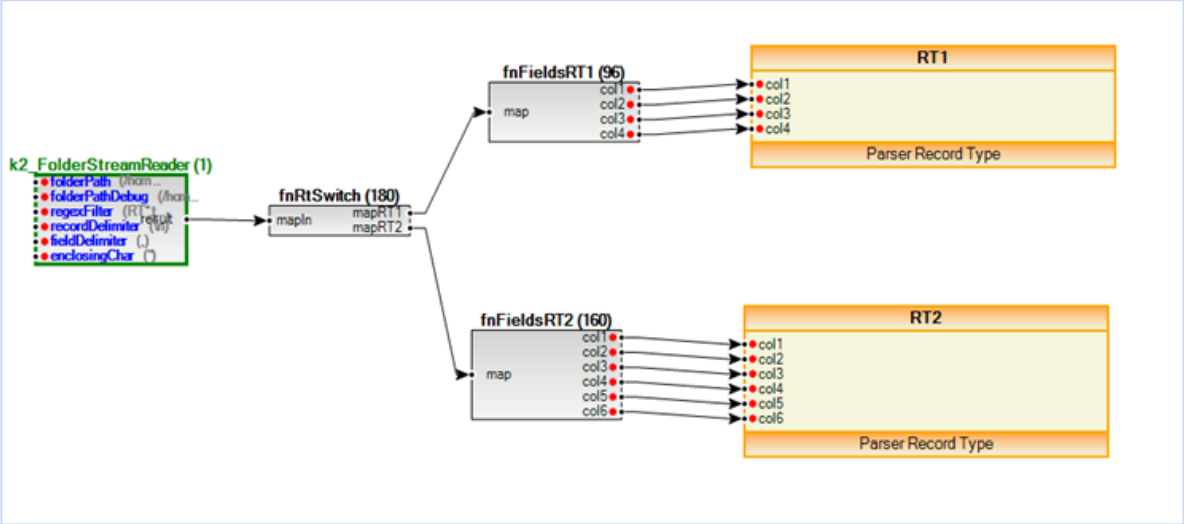


> **Note:** you can create multi record types using new function type "switch function" (enabled only in parser map). This option allows you to define deferent record types for the same parser.

An example of a parser map with SFTP interface type is shown below:



An example of a parser map with cord types is shown below:



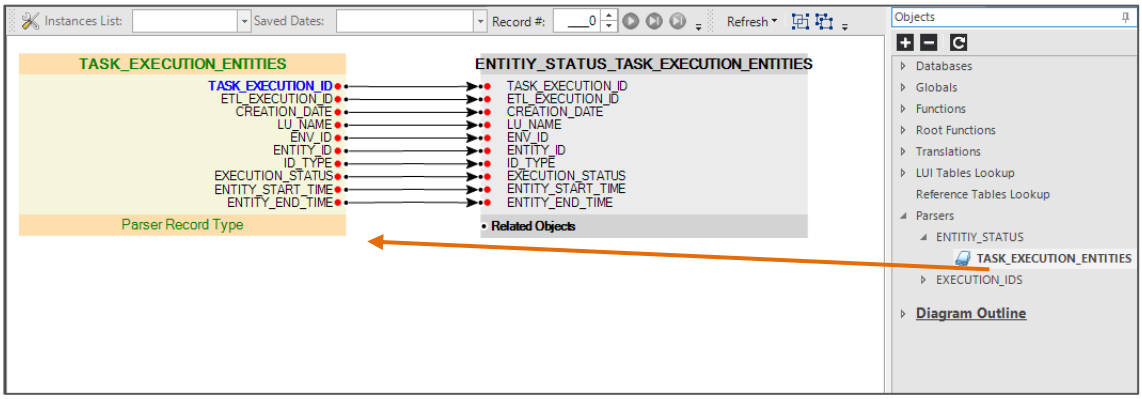
To capture the parsed records in a table

If there is a need to capture the parsed records in a table, create a table under the project tree.

(1) Right-click the required table in the project tree and select a **new table population.**

*The table's **Population** tab displays, including the table's attributes.*

(2) Drag the parser's object from objects pane (appears at the bottom right of the window) into the table's **Population** tab.



The parser's record type is displayed, as shown above.

(3) The following two options are possible for linking between the parser record and the table (both are available by right-clicking the parser object):

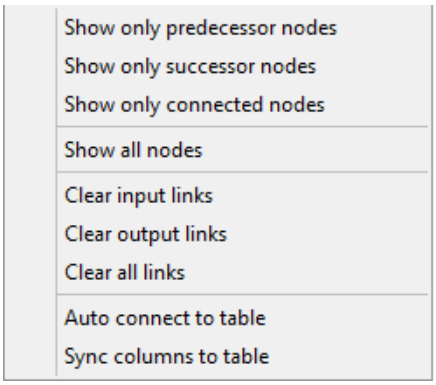
- **Auto connect to table** – Creates automatic mapping links between the parser fields and the table fields that have the exact same names.
- **Sync columns to table** – Automatically creates mapping links between the parser fields and the table's matching field names. It also creates new fields

in the target table for every unmatched field and links them. Specify a name for the table's population at the bottom of the tab.

(4) Click the **Save** icon or press CTRL+S to save your population.

The population map is saved under the project's table in the project tree pane.

(5) During the fields' mapping, additional options can be used (besides the ones mentioned above):

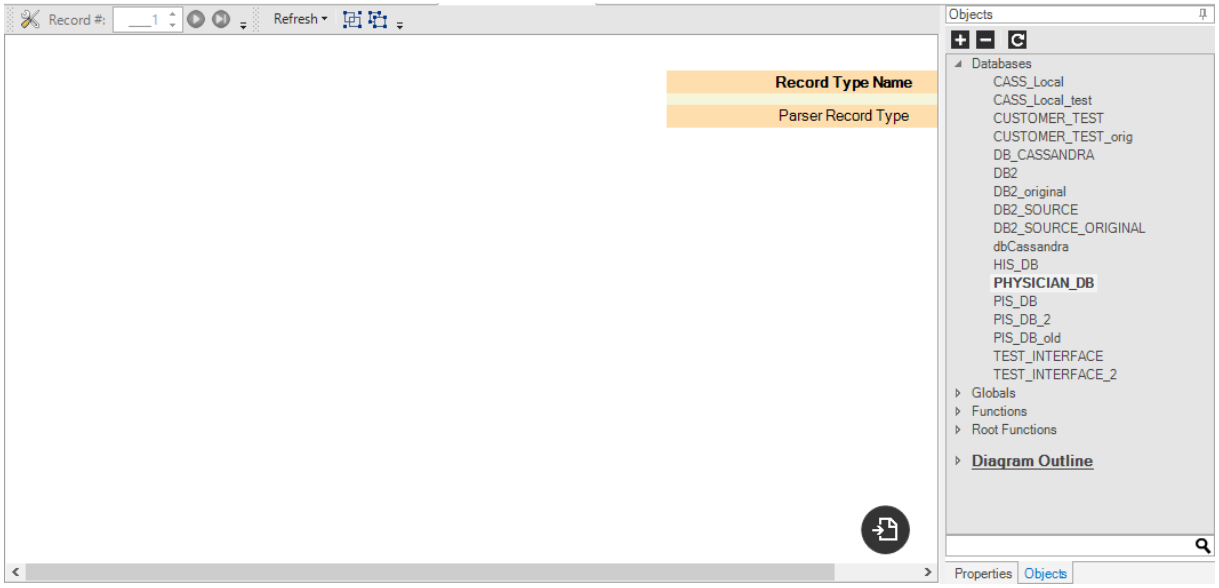


- Show only predecessor nodes, Show only successor nodes, Show only connected nodes, and Show all nodes: see Section 8.12.
- Clear input links, Clear output links, and Clear all links: see Section 8.18.

13.2.4 Support New DB-Query as Root Object for Parsers

You can create a new parser map using DB Query as an input, such as a creation of a table population.

(1) Open a new parser map window:



(2) Select the DB interface name under Databases option in the Objects tab.

(3) Select a table under the selected DB interface and drag the required DB table to your window to create a DB-Query as a root node in the parser map.

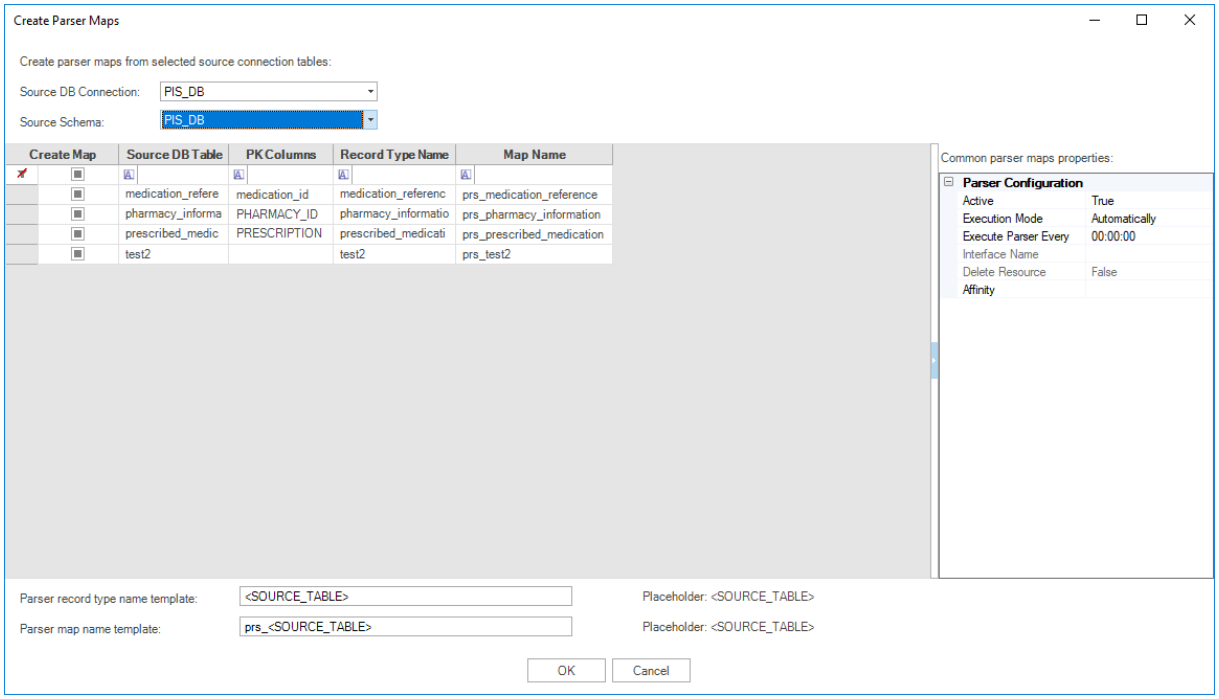
> **Note:** When creating a parser based on DB-Query, the Interface Name and Delete Resources properties in the Parser Configuration section of the parser map are disabled for editing.

13.2.5 Support Creating Multi Parsers Based on DB Tables

This option enables a bulk creation of parsers based on DB tables.

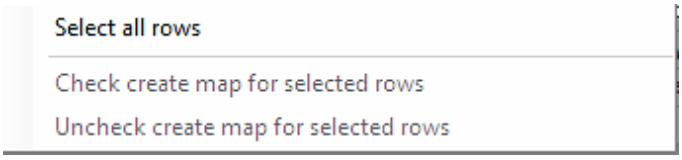
A new context menu command was added on project tree over parsers node: 'Create Parser Maps Based on DB'.

This command opens a new window, allowing you to select DB interface and schema. Then, it populates the grid by all available tables from the selected schema:



(1) Right-click into the Create Parser Maps window.

The Context menu is displayed.

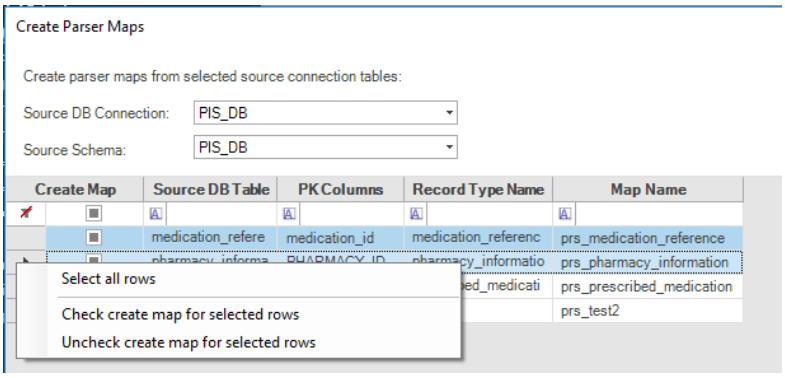


(2) To create a parser:

- For all the rows in the grid: select the 'Select all rows' option, and then right-click and select 'Check create map for selected rows'.

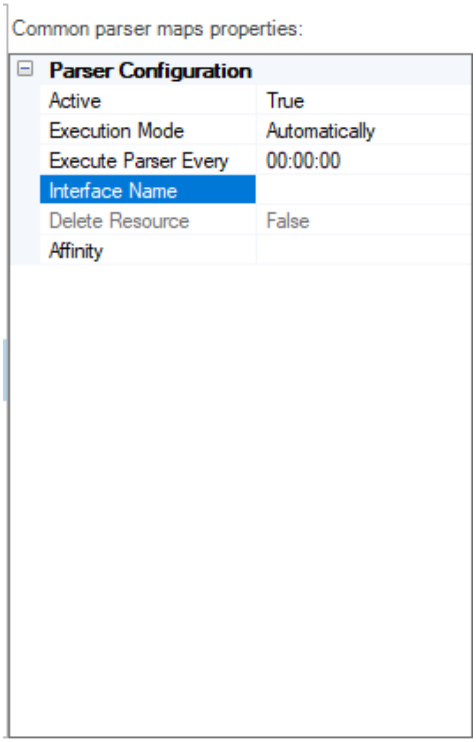
or

- For selected rows in the grid: mark some of the rows, right-click and select the 'Check create map for selected rows' option.



(3) To cancel your selection, mark the required rows, and select the 'Uncheck create map for selected rows' option, or select the 'Create Map' checkbox to deselect it.

(4) Set common parser maps properties. You can set properties for all parser maps, selected by you.



(5) Populate PK (primary key) columns for each selected map.

You can edit the PK columns by using the dropdown list of the tables columns, or edit this field manually.

You can change the order of the PK columns by editing manually the PK columns field.

(6) Click OK. (0)

> **Note:** You can edit the parser record type name and parser map name. However, both names must contain the source table name.
>
> **Note:** PK columns field is mandatory. By default, the PK column is populated based on the PK of the source table. You can edit the PK columns and add or remove columns to the PK.
>
> **Note:** The first column of the PK is the partition key. The next columns of the PK are the clustering key.

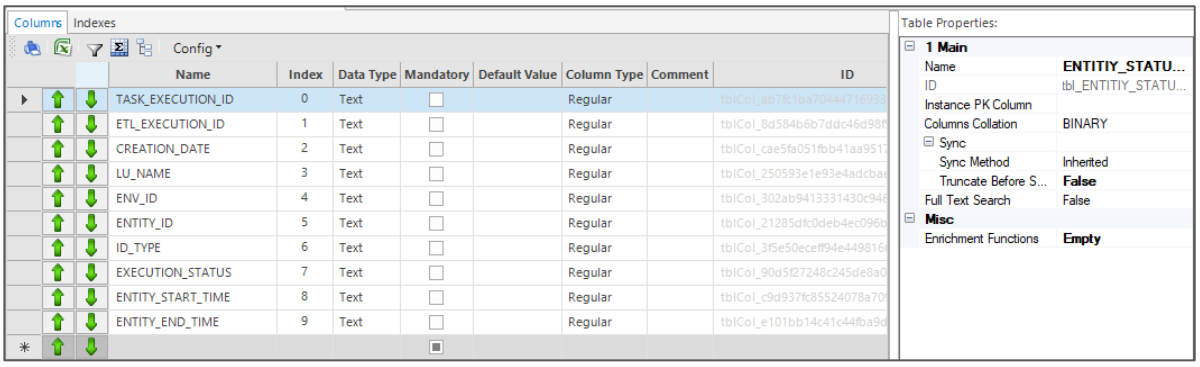
13.2.6 Deleting Parser Records

You can define whether a parser record is to be deleted after it is synchronized into the parser table.

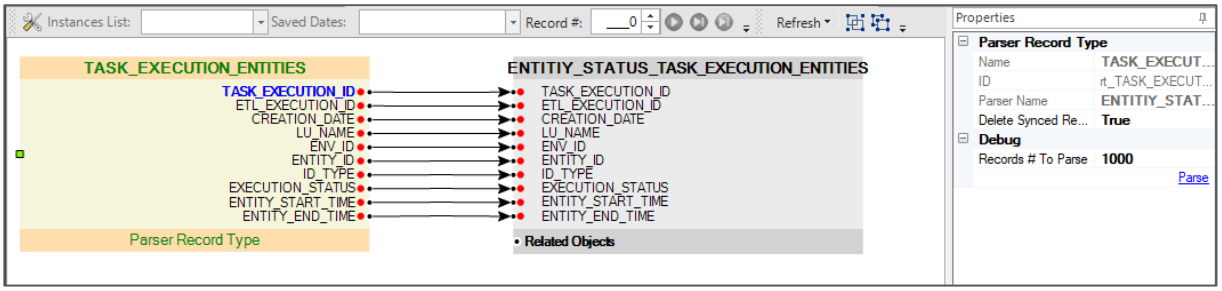
☞ **To delete a parser record after synchronization into the parser table**

(1) Select a table in the project tree.

(2) In the Properties pane, set the **Truncate before Sync** property to **False**.



(3) Open the table population object and click the parser record type in window. Then set the **Delete Synced Records** property to **True** to delete the parser record after it is synced into Fabric.



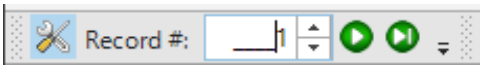
13.2.6.1 Parser Debug

K2View Studio enables you to debug the parser map.

(1) Click the **Wrench** icon to enable debugging and select the record number.

(2) Click the green **Play** button to run the parser debug.

The data streams are displayed next to each map link.



> **Note:** When using the **k2_FolderStreamReader** built-in function, the **fileDebugPath** parameter should be defined. This parameter specifies the input files' folde

13.2.6.2 Table Debug

It is possible to debug a table that is derived from a parser input.

(1) Click the **Wrench** icon to enable and select an instance ID.

(2) Type the instance ID or select an ID from the dropdown list.

(3) In the **Saved Dates** dropdown, select **Generate New File/Newest File** or a specific date when the data was generated.

(4) Click the green **Play** button.

The relevant data record displays in the link lines.

When more than one record appears for a given instance, it is possible to filter and debug a specific record number.

> **Note:** A table that is derived using a parser can be viewed using a Data Viewer, as described in Section 7.12.

13.2.7 Notes

- When you populate the affinity for a parser, Fabric allocates an execution server based on the populated affinity. If the affinity refers to several Fabric nodes- for example- DC name or the Logical node name contain several Fabric nodes- one of those nodes executes the parser.
- When affinity is not populated- Fabric distributes the parsers between different nodes of the Fabric cluster.
- If you run startparser command using cqlsh and set an affinity in your command- this affinity overrides the Studio's definition for your parser. For more information about cqlsh commands for parser, please refer to the Fabric Runtime User Guide document.
- Fabric still supports the old style of parsers for compatibility purpose. The implementer can decide whether to use the old functionality (key column) or the new one for parsers, created by older versions of Fabric Studio. However, once you remove manually the key column and save your changes, the parser is updated to function according to the new functionality. Next time you open this parser, you will not be able to view (and set) the key column any more.
- When linking a population object that uses a parser as an input to a parent table in the LU schema, a new validation will validate that these links are connected to the PK fields of the parser record type in their ascending order.

14 Working with Environments

This feature enables users to define a number of Environments according to the specific needs and to switch between the Environments in the same Fabric. Different users can work in different Environments at the same time, but the performed changes might affect the entire data in Fabric.

14.1 Creating New Global Environment

The created Environment is shared and applied to all Logical Units (LU) existing in the Fabric.

The default environment is called "**_dev**".

☞ To create a new Environment:

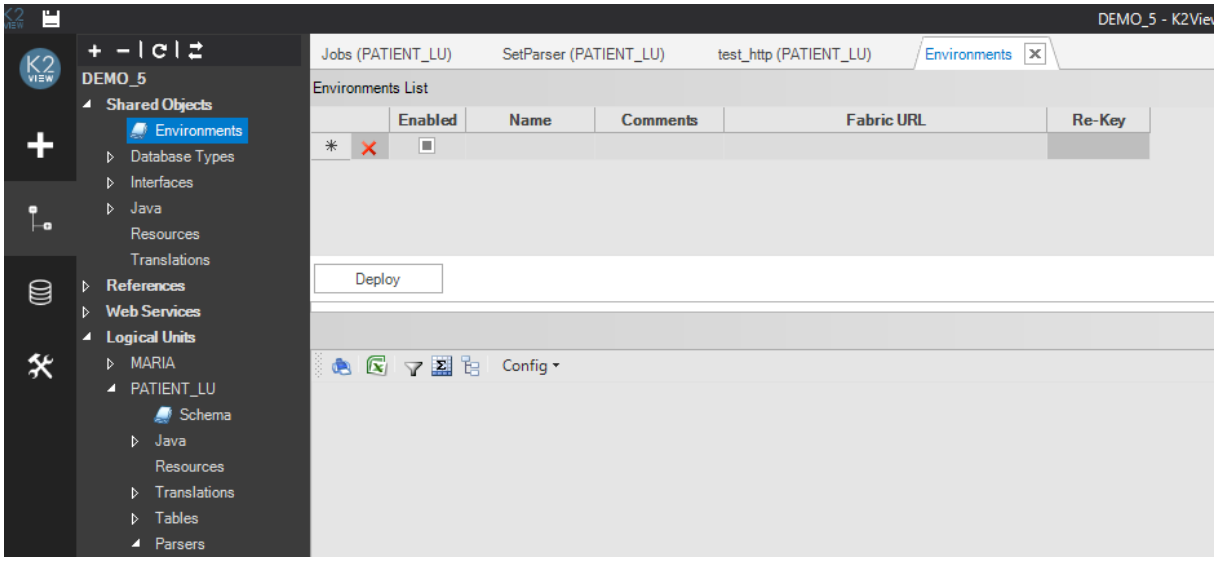
(1) Open the **K2Fabric Studio** application.

(2) Create a new project or open an existing project.

The entire project is loaded.

(3) In the project tree, select **Environments**.

*The **Environments** tab appears in the working area.*



(4) In the name field, enter a meaningful name for the new Environment.

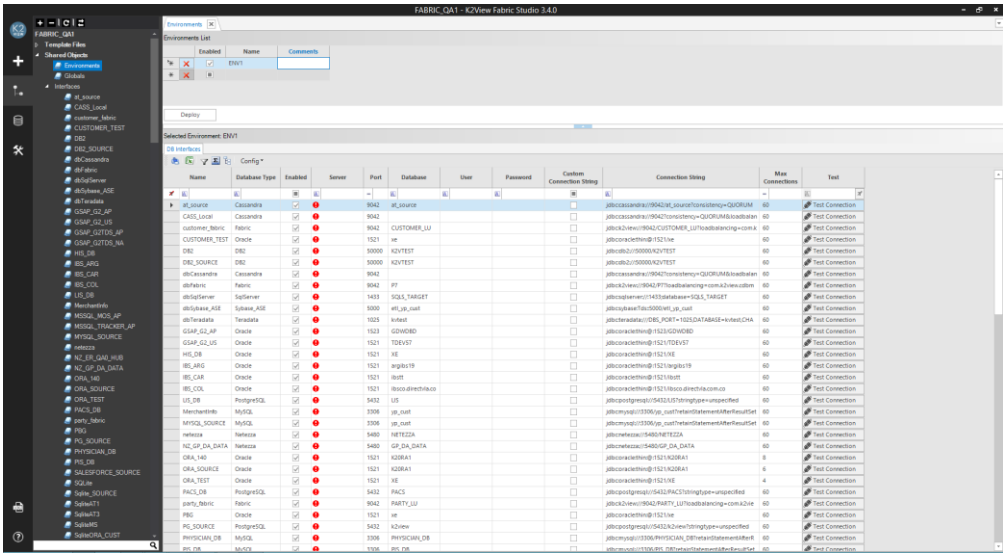
(5) In comments field, enter any comment related to the environment if needed.

(6) In the Fabric URL field, fill in the fabric node to be used for a strong encryption mechanism for all the interface passwords related to the given environment.

(7) Click on Re-key field in case interface passwords should be encrypted again.

Once clicked, the text field becomes enabled and all existing DB interfaces / generic interfaces of the project are copied and displayed in the DB Interfaces/generic interfaces tabs.

By default, all interfaces are enabled.

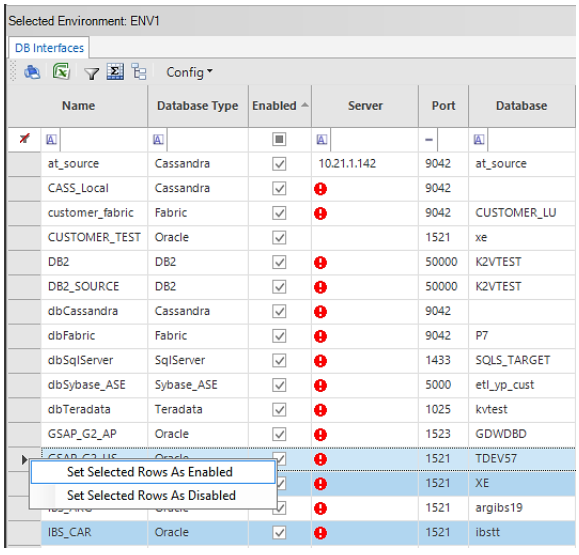


(8) Select the required interfaces, which you wish to add to the newly created environment. To select/deselect interfaces, perform one of the following:

- In the Enabled column, select or deselect the check boxes for the desired interfaces.

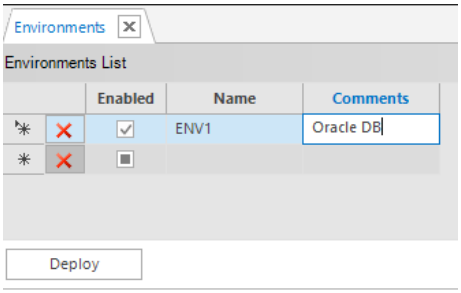
  or

- On the Interfaces tab, right-click a row and select **Set Selected Rows as Disabled** or **Set Selected Rows As Enabled.**

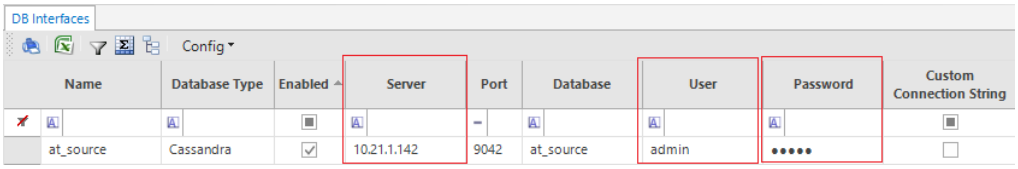


> **Tip:** You can use the Ctrl key for multiple selection.

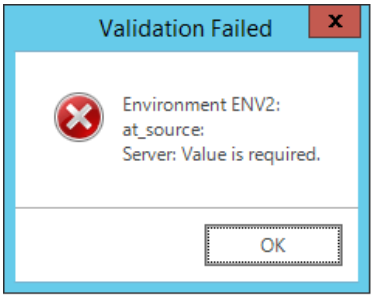
(9) In the **Comments** field, enter a meaningful comment if required. This field is not mandatory.



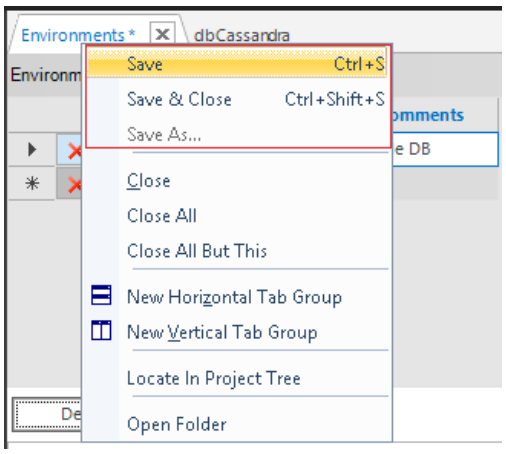
(10) Fill in the relevant fields for each active interface for the new Environment. See the example in the below picture.



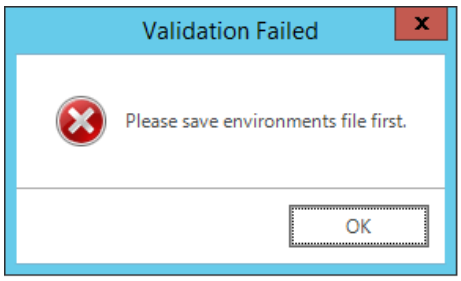
Make sure you enter all mandatory values (indicated by), otherwise the new Environment cannot be saved. A message appears, notifying that the values are missing.



(11) Right-click the **Environments** tab and select one of the available save options to save the created Environment.



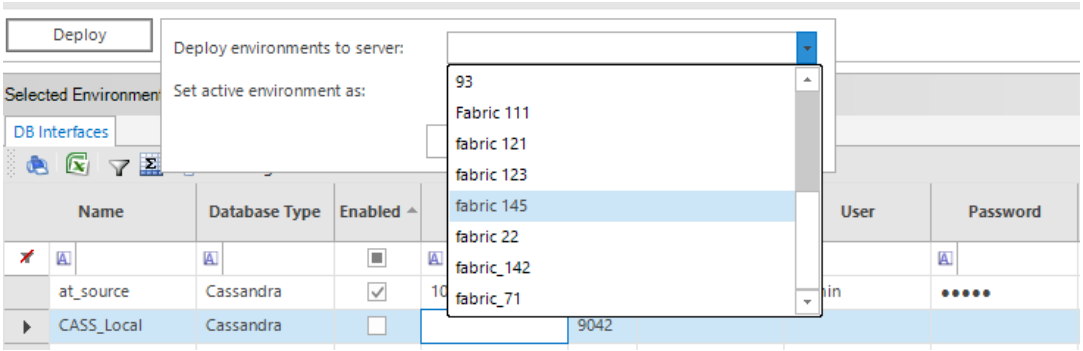
Note: If you click Deploy, before saving the created Environment, the **Validation Failed** message appears asking to save the Environment. Click **OK** to close the dialog box and save the Environment.



14.2 Deploying Environment to Server from Studio

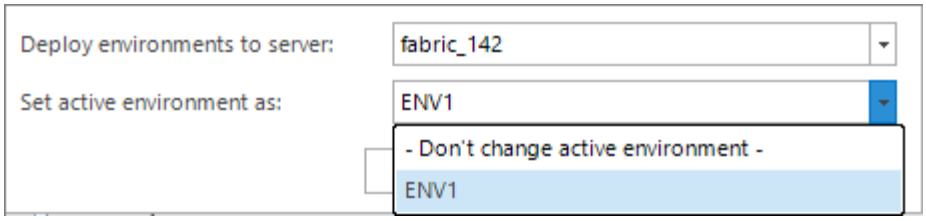
(1) Open an existing project and from the project tree, select **Environments**.

(2) Click **Deploy** to open the actions list, and select a Fabric to which to deploy the DB interfaces.

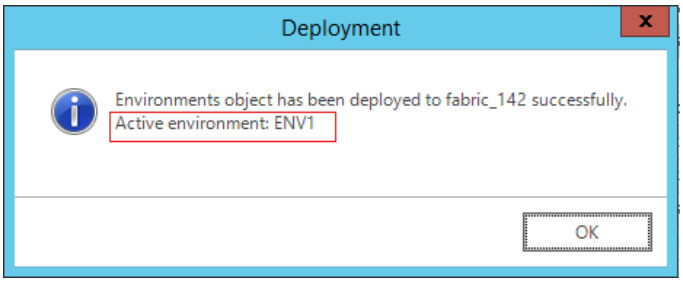


(3) In the **Environments** tab, click **Deploy** to open the actions list and select **Set active environment as**:

(4) Select the available environment.



*The following **Deployment** dialog box appears, showing a currently active Environment.*



(5) Click **OK** to close the dialog box.

14.3 Manually Deploying Environment to Server from XML File:

It is possible to deploy an Environment to a server from an xml file that contains all interfaces with all existing environments.

☞ To deploy Environment to a Server from an XML file:

(1) Connect to a server by using the command `cqlsh`.

(2) Use the command: `Deploy environments from file '{filename}'`

> **Note:** Deploying an xml overrides all existing environments, except for the "_dev", which is the default environment.
>
> If the environment exists in Fabric, but not in the deployed xml, it is removed from Fabric.

Fabric encrypts the passwords in the file (if not already encrypted) and saves the xml with the encrypted passwords.

The required permission for this action: DEPLOY_ENVIRONMENTS.

Example of the xml:

<?xml version="1.0" encoding="utf-8"?>

```
<FabEnvironmentsManager xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <Name>Environments</Name>
 <EnvironmentsList>
  <Environment enabled="true" name="ENV1">
   <Comments>Oracle DB</Comments>
   <DbInterfacesList>
    <DbInterface k2StudioObjectName="at_source" dbType="cassandra" active="true">
     <dbHost>10.21.1.142</dbHost>
     <dbPort>9042</dbPort>
     <dbScheme>at_source</dbScheme>
     <dbUser>admin</dbUser>

<dbPasswordEncrypted>MaOKTRCpBHfN1xKF0rSrxd38K4Op94xaQWVk1TRHPvg=:P7Y/n1VCwn1dAqp/TO3v9XA1MHoTFL3RXEXiqxe3fcY=</dbPasswordEncrypted>

<connString>jdbc:cassandra://10.21.1.142:9042/at_source?consistency=QUORUM&amp;loadbalancing=com.k2view.cdbms.policy.SingleNodePolicy('10.21.1.142')</connString>
     <maxConnectionsNumber>60</maxConnectionsNumber>
    </DbInterface>
    <DbInterface k2StudioObjectName="CASS_Local" dbType="cassandra" active="false">
     <dbPort>9042</dbPort>
     <dbScheme />

<connString>jdbc:cassandra://:9042?consistency=QUORUM&amp;loadbalancing=com.k2view.cdbms.policy.SingleNodePolicy('')</connString>
     <maxConnectionsNumber>60</maxConnectionsNumber>
    </DbInterface>
   </DbInterfacesList>
   <GenericInterfacesList />
  </Environment>
 </EnvironmentsList>
</FabEnvironmentsManager>
```

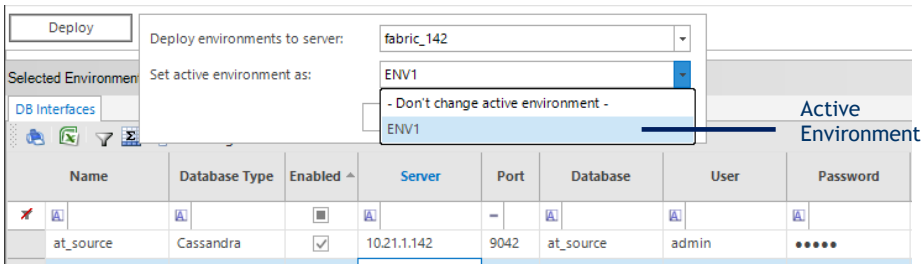
> **Tip:** You can check all exisitng global Environments by using a command in Fabric (see Fabric runtime document).

14.4 Switching Between Global Environments

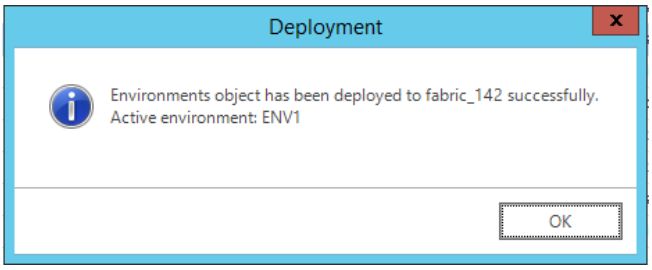
14.4.1 Method 1

(1) In the **Environments** tab, click **Deploy** to open the actions list and select **Set active environment as**:

(2) Select the Environment from the list of the existing Environments.



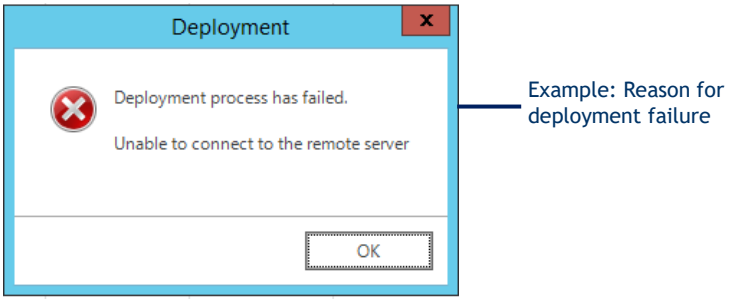
At the end of the deployment, the following message appears.



> **Note:** Active Environment affects all Fabric servers in the cluster, i.e., once changed, the new Environment becomes an Active Environment for all Fabric servers in the cluster.

(3) Click **OK** to close the dialog box.

If the deployment fails, the following dialog box appears indicating the deployment failure and the reason for failure.



(4) Click **OK** to close the dialog box.

(5) Fix the reason for failure and repeat steps 1-3 above.

14.5 Checking Active Environment

14.5.1 Method 1

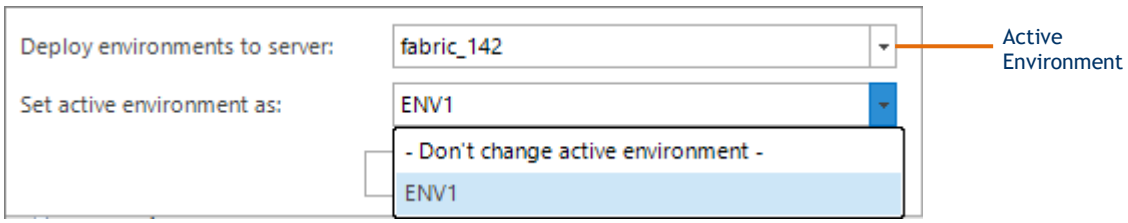
In User Utils, add the method:
`'Get Active Environment Name'`

14.5.2 Method 2

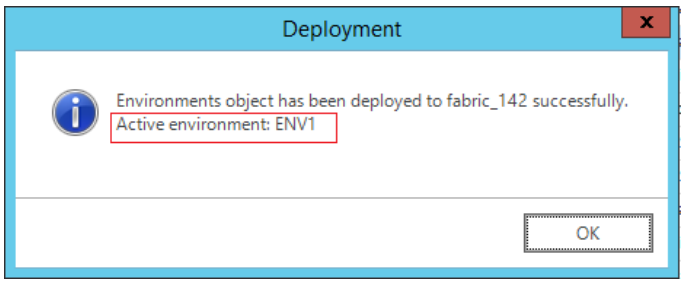
(1) Open an existing project and from the project tree, select **Environments**.

(2) On the **Environments** tab, click **Deploy** to open the actions list and select **Set active environment as**:

(3) Select **Don't Change Active Environment**.



The following **Deployment** dialog box appears, showing currently active Environment.



(4) Click **OK** to close the dialog box.

15 Web Services

K2View Fabric clients can access the K2View Fabric data using web-services that query the entire universe of data and present the relevant data. The web services should be deployed to the K2View Fabric separately. These can then be invoked either by the clients or directly from any web browser.

K2View Fabric Studio supports defining a web service function, saving it to the project file, and deploying it to the K2View Fabric Server.

A web service is defined as a function.

Functions useful for the Web Service:

- ci.execute (Db ci) - gaining access to a specific schema and instance id. See more information in Section 15.
- fabric().fetch- get results from a given interface by SQL query and optional list of parameters.

The following sections explain the use of K2View Fabric Studio to create an appropriate function as a Web Service and deploy it to the K2View Fabric Server.

A prerequisite for automatically generating code for a Web Service in K2View Fabric Studio is the existence of a data file for the LU Type. To create the data file, see Section 7.12 on **Data Viewer**. Remember that by viewing the data, K2View Fabric Studio generates a data file for the Instance supplied during that process.

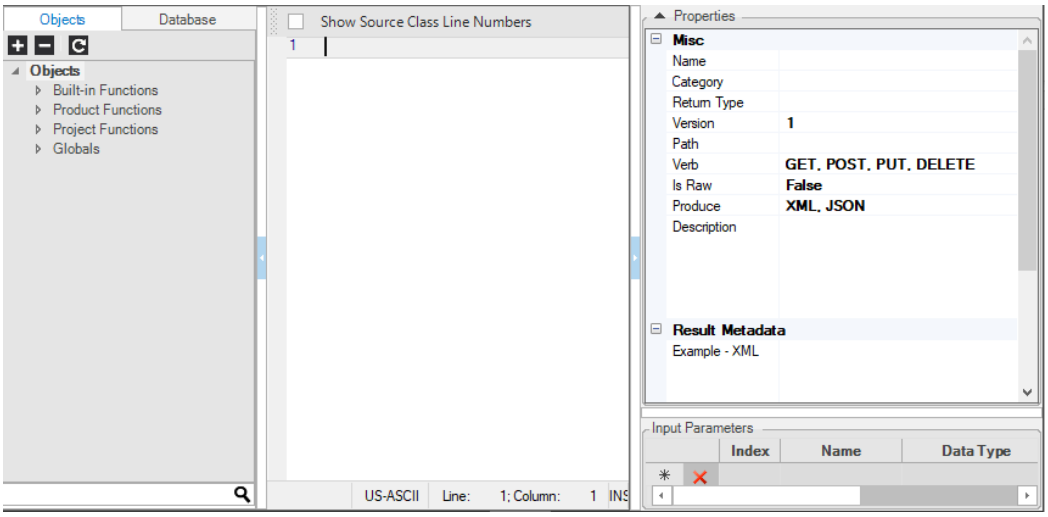
For the full Fabric REST API specification, please refer to Fabric REST API Specification document

15.1 Creating a Web Service

☞ To create a new web service

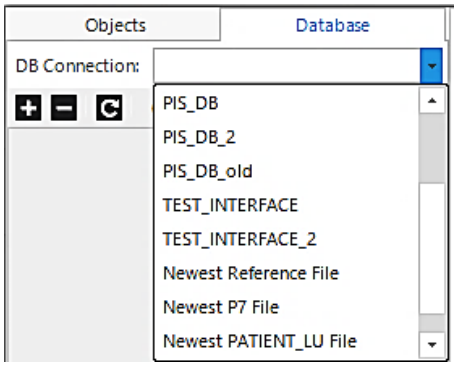
(1) Right-click the Web Services in your project tree and click **New Web Service**.

The Function Manager window is displayed.



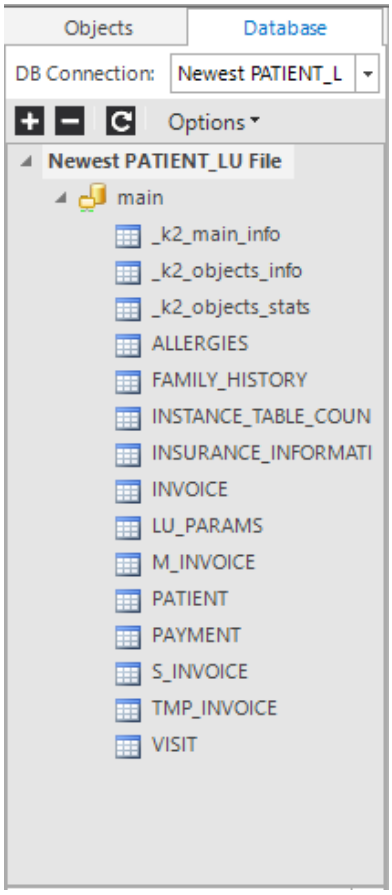
(2) Click the **Database** tab in the Objects/Database pane.

(3) Select from the dropdown list a DB interface, newest reference file, or newest LU file.



(4) Select the required DB option in the DB Connection field. If you need to access fabric, you need to select the **Newest <LUname>** .

The Database pane displays the data file structure.



(5) Click the main data file indicator in the data file structure.

The list of child entities is displayed.

(6) Click a table name for the primary table to retrieve data.

The list of data columns is displayed.

(7) Select the instance identifier column.

(8) Right-click the selection.

The Context menu is displayed.

(9) Select **Add selected to input parameters** option.

The column name, data type, Mandatory and comment appear in the Input Parameters list (lower right) pane.

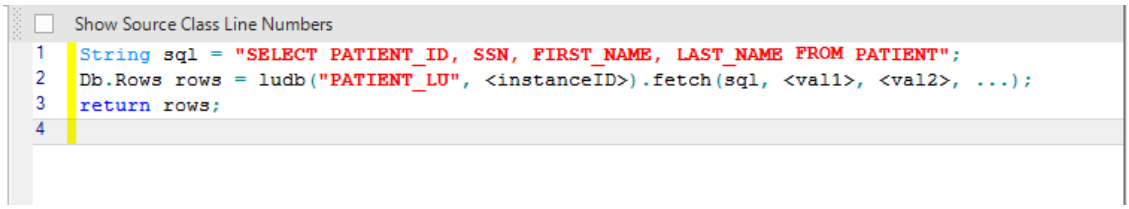
(10) Select columns to be retrieved from the table. Multiple columns may be selected by pressing the *Ctrl* key.

(11) Right-click the selection.

The Context menu is displayed.

(12) Write the required code or use the selected **Generate code** option.

Generated code is displayed in the Java editing pane of the window. The generated code has the basic structure that includes the call to the interface or ludb and the fetch method.



> Note: you can still use the DBExecute function, followed by the declaration of a SELECT statement and invocation of the DBQuery function in your WS. See example below:

ci.execute("get PATIENT_LU." + <instanceID>, null);

String sql = "SELECT PATIENT_ID, SSN, FIRST_NAME, LAST_NAME FROM PATIENT";

Object[] valueArr={};

Db.Rows rs = fabric().fetch(sql, valueArr);

return rs;

(13) Edit the code as described in Section 0.

(14) Enter a name of the Web Service in the **File name** field (window footer).

(15) Select a category for the Web Service in the **Category** field (window footer).

(16) Select a Web Service return type in the **Return Type** field (window footer).

(17) Set the Web Service versioning in the **Version** field. By default, it is populated by 1. The purpose of version control is to give users some ability to keep their current integrations until they can complete any changes required to handle the newest version. This way, you can define several Web Service versions, related to the same service. For example, wsGetCustomerInfo has version 1 and wsGetCustomerInfoDev has version 2.

(18) Set the URL path for the Web Service in the **Path** field in order to support versioning for the same service. The path is populated by the method name, populated in the URL and several web services can have the same Path value. For example:

(19) wsGetCustomerInfo WS has version 1 and path 'getCustomerInfo'.

(20) wsGetCustomerInfo WS has version 2 and path 'getCustomerInfo'.

(21) When you run the following URL- you invoke wsGetCustomerInfo WS-

(22) http://<Fabric IP address>:3213/api/v1/getCustomerInfo?token=t1&format=json&customerId=543;

(23) When you change the version of the URL from v1 to v2- you invoke wsGetCustomerInfoDev.

(24) Set the Verbs to be supported by the Web Service in the **Verb** field (supporting: GET, POST, PUT and DELETE).

(25) Set the Is Raw field to True or False (by default, it is 'False'). This indicator indicates if the output structure should be manipulated by Fabric automatic mechanism. If this indicator is 'True', then Fabric brings the output data "as is" without any manipulations.

(26) Set the support output format in the **Produce** field. Supported formats: Json (default), XML and CSV.

(27) Set the Web Service description to be present on Swagger in the **Description** field.

(28) Set the **Example-XML, example-JSON and example-CSV** to be presented on the Swagger, before the Web-Service call.

(29) Click the **Save** icon or press CTRL+S to save your changes.

The Web Service is added to the Web Services folder in the Project Tree pane.

15.2 Basic Structure of the Web Service Function

When K2View Fabric Studio automatically generates Java code, it has a set template that includes the following lines of code:

Declaration of a String that holds the SELECT statement, for the <column names> of interest from the <table name> (or Join of tables) of interest.

```
String sql = "SELECT <column names> FROM <table name>";
```

Get entity and fetch records from DB interface by the SQL query:

Get data from Fabric:

```
Db.Rows rows = ludb("PATIENT_LU", <instanceID>).fetch(sql, <val1>,
<val2>, ...);
```

Get data from other DB interfaces:

```
String sql = "SELECT SSN, PHARMACY_ID, PHARMACY_NAME FROM
pharmacy_information";
Db.Rows rows = db("PIS_DB").fetch(sql, <val1>, <val2>, ...);
```

Return statement.

```
return rs;
```

15.3 Editing the Web Service Code

The generated Web Service code provides basic functionality to perform the desired data retrieval. However, quite often there is a need to enhance the code in various ways.

The code is edited using the Java Editor (see Section 17.1) capabilities provided by the code pane of the Function Manager window.

You must replace the <instanceID> listed in the ludb invocation with the web-service input parameter (declared in Step 8 of the procedure in Section 15.1.) that identifies the K2View Fabric instance.

In addition you must populate your SQL query parameters in the place holders for parameters (<val1>, <val2>,…) or delete the replace holders from your code.

For example-

```
Db.Rows rows = ludb("PATIENT_LU", 1).fetch(sql, “XX”, 123);
Db.Rows rows = ludb("PATIENT_LU", 1).fetch(sql);
```

These steps are necessary for a clean compilation of the web service code.

The following are additional points that may need editing:

Edit the SQL statement for various enhancements:

- Add a WHERE clause.
- JOIN additional tables to the query.
- Add advanced features to the SELECT statement, for example assigning identifiers to the tables and specifying which table's columns to retrieve.

Add any additional Java code to the function. The code of the web service can also reference any transformation rules (functions or translations) or Globals defined in the project.

> **Note:** Any K2View Fabric run-time side command (Listed in K2View Fabric Run-Time document) can be used within web service using the execute function.

15.4 Deploying the Web Services

☞ To deploy the Web Services to the K2View Fabric Server

(1) Right-click wither the Web Services, selected web services or categories of the web services in the Project Tree pane.

The Context menu is displayed.

(2) Click the **Deploy to Server** option.

The list of the configured K2View Fabric Servers is displayed.

(3) Click the K2View Fabric Server to deploy to.

(4) The available deployment options are:

- Deploy all web services
- Deploy selected list of web services
- Deploy select list of categories- all the WS, related to the selected categories, are deployed

15.5 Invoking the Web Service

The Web Services are designed to be used by client applications or K2View Fabric data consumers (for example, the customer service desk employees). For testing purposes, K2View Fabric Studio supports invocation of the Web Service and Graphit file directly from within the development environment. By default, the web services are invoked using the Swagger, as described on page 178.

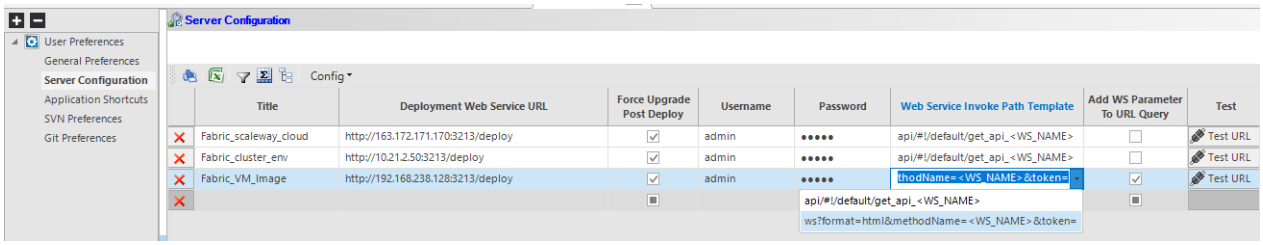
To invoke a web service

(1) The web services or Graphit file are invoked using the Swagger.

(2) Open the **User preferences - Server** configuration.

(3) Select your record and open the dropdown list for a "Web service invoke path template".

(4) Change the template, if needed, and click the **Save** icon.



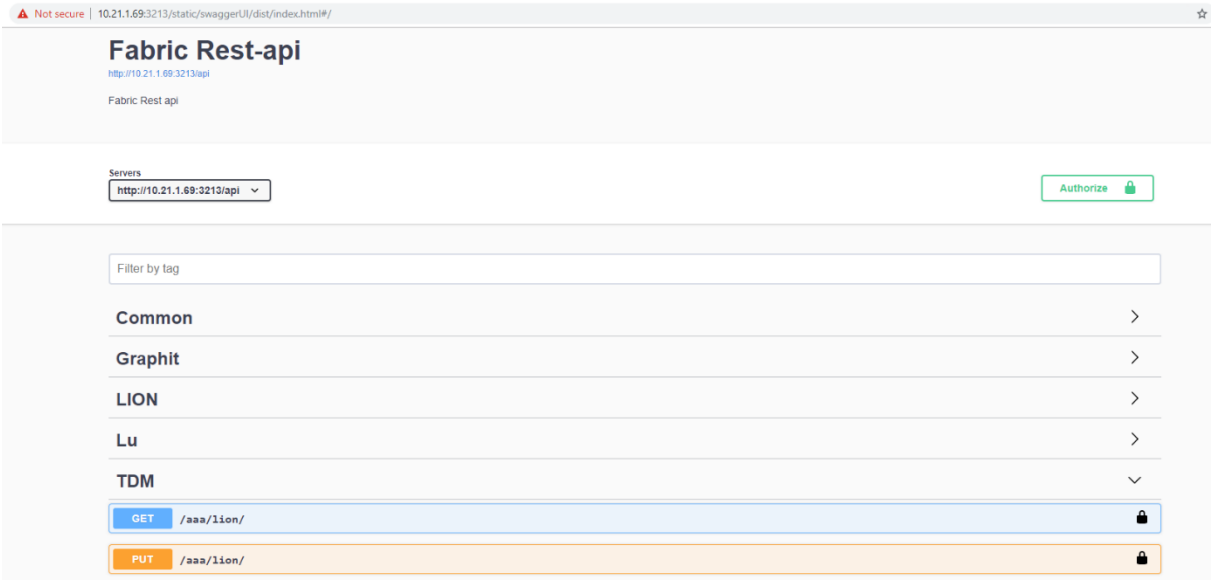
15.6 Invoking and Testing Web Services Using Swagger

K2View Fabric Studio supports invocation and testing of web services using Swagger. Swagger is a specification and complete framework implementation for describing, producing, consuming, and visualizing RESTful web services.

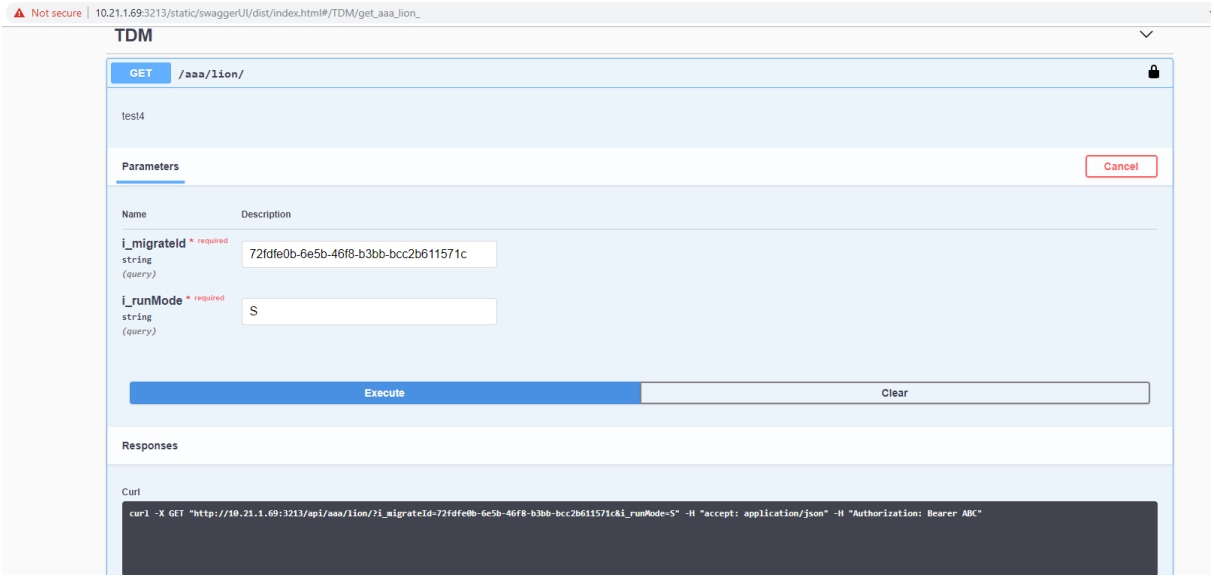
To connect to Swagger

(1) Navigate to the following URL and specify the IP address of your Fabric server: http://<Fabric server>:3213/api or set Web service invoke path template in your U**ser preferences - Server** configuration to static/swaggerUI/dist/index.html#/<CATEGORY>/<WS_VERB>_<WS_PATH>

and then invoke web service (right click the Web Service name in the Project Tree pane and click the Invoke Web Service option).

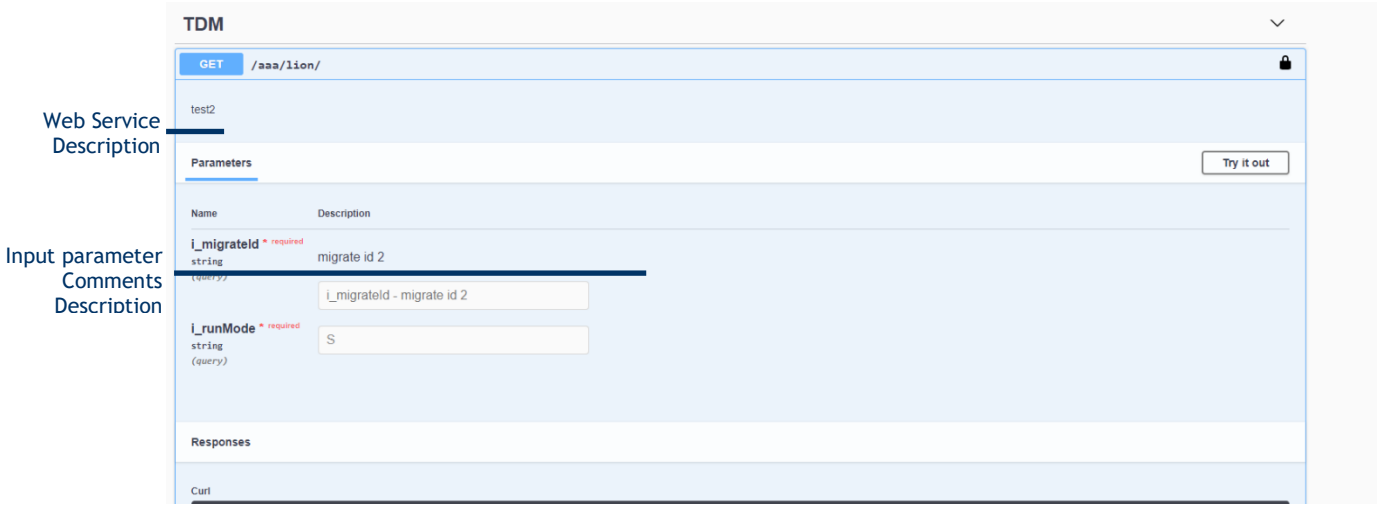


(2) Click a web service in the Swagger user interface to view more details about it, including its parameters, as shown below.

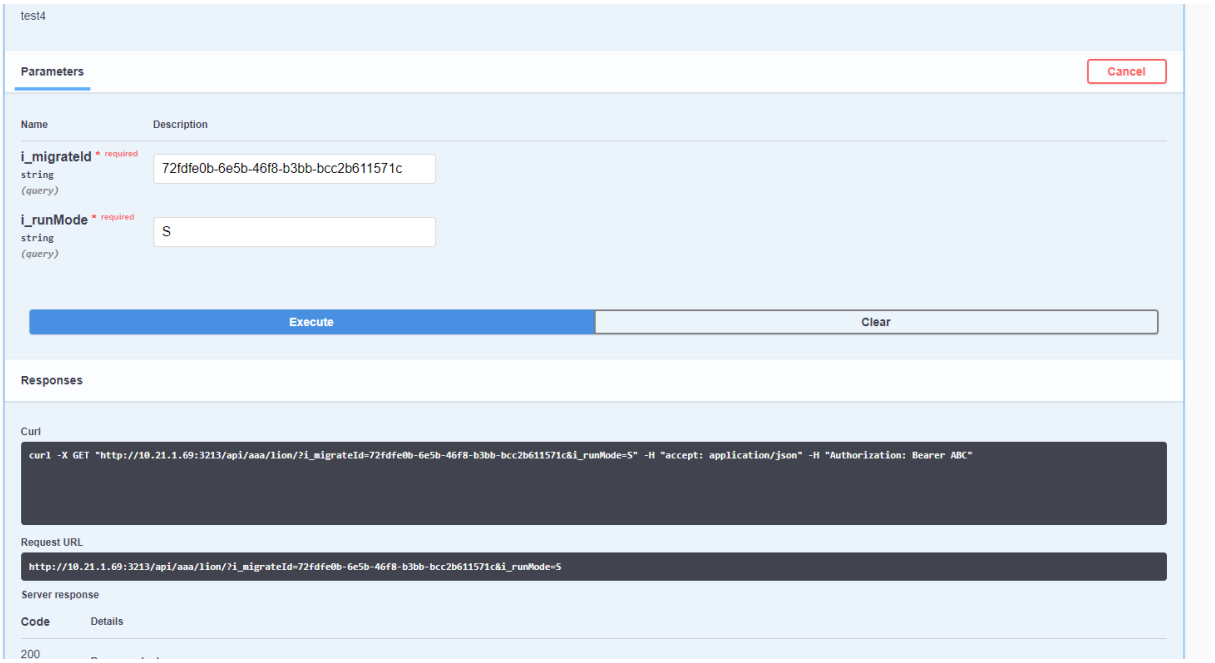


The **authorization Token** and **Media type** fields are always displayed, and are described on page 178. Additional fields may display too, depending on the web service.

> **Note:** If a description of the method or comments were added in K2View Studio, they display in the Swagger user interface, as shown in the examples below:



(3) Fill in the parameters and then click the **Try it out**! button to display a response.




Server response
Code
Details
200
Response body
[
{
"Node id": "172.26.0.102",
"Instance ID": "_dev_3",
"Status": "ADDED",
"Error": ""
},
{
"Node id": "172.26.0.102",
"Instance ID": "_dev_4",
"Status": "ADDED",
"Error": ""
},
{
"Node id": "172.26.0.102",
"Instance ID": "_dev_2",
"Status": "ADDED",
"Error": ""
},
{
"Node id": "172.26.0.102",
"Instance ID": "_dev_1",
"Status": "UPDATED",
"Error": ""
}
]
Download
Response headers
allow: GET, POST, PUT, DELETE
content-length: 315
content-type: application/json
date: Thu, 17 Oct 2019 13:53:15 GMT
Responses
Code
Description
Links
200
OK
No links
Media type
application/json
Controls Accept header.

16 Reference Tables

K2View Fabric projects support defining tables from the source DB as Reference Tables. These tables can be used across the Logical Units (see Chapter 7) within the project. A Reference Table typically contains meta-data that is referenced both across different instances of a particular LU, as well as by instances from different LU's within the project. A classic example of the reference information would be a Postal-code table that can be used to identify the Postal code for customer addresses of many different applications.

16.1.1 Adding a Reference Table

☞ **To add a new Reference Table to the project**

(1) Right click the **References** item in the Project Tree pane of the main window.

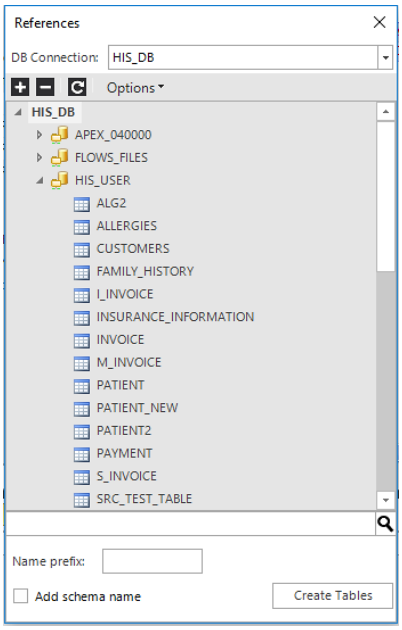
The Context menu is displayed.

(2) Click the **Create References Based On DB Tables** option.

The References popup window is displayed.

(3) Select the **Data Source Interface** from the DB Connection drop-down menu (at the top of the window).

The Directory of tables in the source DB is displayed in the window.



(4) Select a table to add as a Reference Table.

> **Note:** Multiple tables may be selected by pressing the Ctrl key.

(5) If required (see a recommendation below) enter the prefix for the Reference table name in the **File name prefix** field (in the footer of the window).

> **Recommendation:** If there is more than one object in the project with the same name, you can set a prefix to be added to differentiate between Reference tables and LU tables that have similar names. We recommend to use a prefix that indicates the project name and that this is a Reference table. For example <proj-prefix>_REF.

(6) Click the **Create Tables** button.

The New Reference table appears in Project Tree under References.

> **Note:** If a new table does not appear immediately, click Refresh icon ( ) at the top of the Project Tree and a new table appears.

16.1.2 Editing and Viewing Reference Tables

Reference tables may be edited in many ways. You can change the default data mapping, add transformations, and/or add or remove columns in the same manner as LU tables. For more information on these procedures, see Chapter 8.

Viewing the table data is available from the Data Viewer window described in Section 7.12.

☞ To view the Reference Table data

(1) Locate your mouse on your References option in the project tree. An icon of data viewer will be displayed near the References option. Then click on this icon to open the data viewer for your reference tables

The Data viewer window is displayed with data file hierarchy in the Instances Tree pane.

(2) Click the data file to display.

The Hierarchy of components is displayed in the Instance DB Tree pane.

(3) Click the table name.

The Table data is displayed in the main Data Viewer window.

16.1.2.1 Set Background Sync for Reference Tables

In the previous versions, the reference tables were synced "on-demand". When operation that involves accessing data from a reference table took place, Fabric checked if this table should be synced according to the sync policy, and refreshed data from the source system as needed.

From Fabric 5.0 and on, reference sync is done in background by default for each table according to sync policy.

16.1.3 Accessing the Reference Table from LU Schema

Before accessing the Reference Table from a specific LU, the Reference Table must be configured.

☞ **To Configure LU to use a Reference Table**

(1) Open the **LU Schema Window** as described in Section 7.6.1.

The schema is displayed with right panel displaying.

(2) Click the **References** tab in the right panel.

The Check list of all project reference tables is displayed.

(3) Select boxes for relevant reference tables.

(4) Click the **Save** icon or press CTRL+S to save the association.

> **Note:** Accessing reference tables can be performed through lookups, web-services, or functions.

16.1.4 Deploying Reference Tables

☞ **To deploy the Reference Tables.**

(1) Right-click the **References** item in the Project Tree pane.

The Context menu is displayed.

(2) Select the **Deploy to Server** option.

The menu of configured server scripts is displayed.

(3) Select the Server URL to deploy to.

Studio begins execution of the deployment and displays a Message Box informing whether the deployment was successful or failed.

17 Other Studio Capabilities

17.1 Java Editor

The Function Manager window, used to create and edit Function transformations (Section 9.2) and Web Services (Chapter 15), includes a Java code editor pane. The editor supports all basic editing capabilities.

The editor is integrated with the K2View Fabric environment and the project structure. In addition, the editor is context-sensitive, supporting Java syntax and functions. The following subsections detail these additional advanced features.

17.1.1 Support for K2View Fabric Elements

The Java editor includes a pane on the left that displays the Object tree of the current project. The object tree includes entities for the Functions – Built-in functions, Project functions, and Product functions – and the Globals defined within K2View Fabric Studio.

In addition, the pane displays the Database entities – tables and columns – under the Database tab.

☞ **To insert a K2View Fabric function into the Java function**

(1) Display the elements in the Objects/Database pane.

(2) Double-click the function or global to insert the element into the editor window.

The Editor displays the function call with a hint of what parameters to supply.

☞ **To insert a SELECT statement for columns in Database table**

(1) Click the **Database** tab.

(2) Select the DB Interface or LU data file in the **DB Connection** field.

The Objects pane displays a list of tables.

(3) Click a table name.

The Objects pane displays a list of columns in the table.

(4) Select columns to use in the SELECT statement (Multiple selection with Ctrl key).

The Editor generates a SELECT statement and displays it in the editor pane.

17.1.2 Support for Java Elements

All Java language reserved words and data types used in the editor pane are displayed in blue. In addition, when a Java built-in function such as “trim” or “toUpperCase” is entered, the editor displays a hint of what parameters are expected.

17.1.3 Keyboard Shortcuts and Auto Completion

Java editor supports the following keyboard shortcuts:

Ctrl-S	Save the changes.
Ctrl-F	Search editor window for string.
Ctrl-Space	Complete name of a Function or Global. When used with Java statement that has different options - presents options (see example below).
Tab	Complete syntax for a Java statement (see example below).

Examples of shortcut uses:

Built-in function name completion

Type “DB” and press the **Ctrl-Space** key.

The Editor displays a list of all server built-in functions.

Type “k2_” and press the **Ctrl-Space** key.

The Editor displays a list of all product functions

Syntax completion

Type “try” and press the **Tab** key

The Editor displays the following syntax:

```
try {
}
catch (expression) {
   throw;
}
```

Syntax Selection List

(1) Type "for" and press **Ctrl-Space**.

The Editor displays a selection list with the following choices:

for

for DBQuery

for yield

(2) Select one of the choices.

(3) Press the **Tab** key. (0)

The Editor displays the following syntax based on option selection:

for	`for (int i=0; i < length; i++) {` `}`
for DBQuery	`for (Object[] row : rs) {` `}`

for yield	`for (Object[] row : rs) {`
	`  yield(row);`
	`}`

17.1.4 Java Class Method Completion

When using a Java core class, the editor presents the options to allow easy insertion of the class and methods:

(1) Type "java."

The Editor displays list of Java core classes.

(2) Select class.

(3) Type "." (0)

The Editor displays a list of methods defined for a class.

17.1.5 Opening Query Builder

☞ **To open the query builder from the Java Editor**

(1) Right-click a query that appears in the editor pane.

The Context menu is displayed.

(2) Select the **Open Query Builder** option.

The second-level context menu is displayed.

(3) Select the **Schema** option.

The Query Builder is displayed in a new window. See Section 17.2 for more information on using the Query Builder.

(4) When finished, click **OK** to save changes to the function.

17.2 Query Builder

The Query Builder is an embedded visual query building component that allows you to build complex SQL queries using an intuitive interface. Basic knowledge of SQL concepts is necessary to use the Query Builder effectively to create the correct SQL code.

17.2.1 Query Builder Window

The Query Builder window displays two tabs that affect what is displayed in the window.

The Query (schema) tab

When the Query tab is selected, the Query Builder window displays:

- DB Tree display pane at the top left – displays the tree of tables and columns for the DB selected from the drop-down list of configured DB Connection Interfaces.
- Main window at the top right a graphic map of the selected tables from the DB and the columns within the table that are relevant to the query.
- Table of selected columns on the bottom right – presents the same information as the graphic map but in a tabular form.
- Query display along the bottom of the window – presents the actual SQL statement that is being generated. This statement may be manually edited to arrive at the final desired form.

Result tab

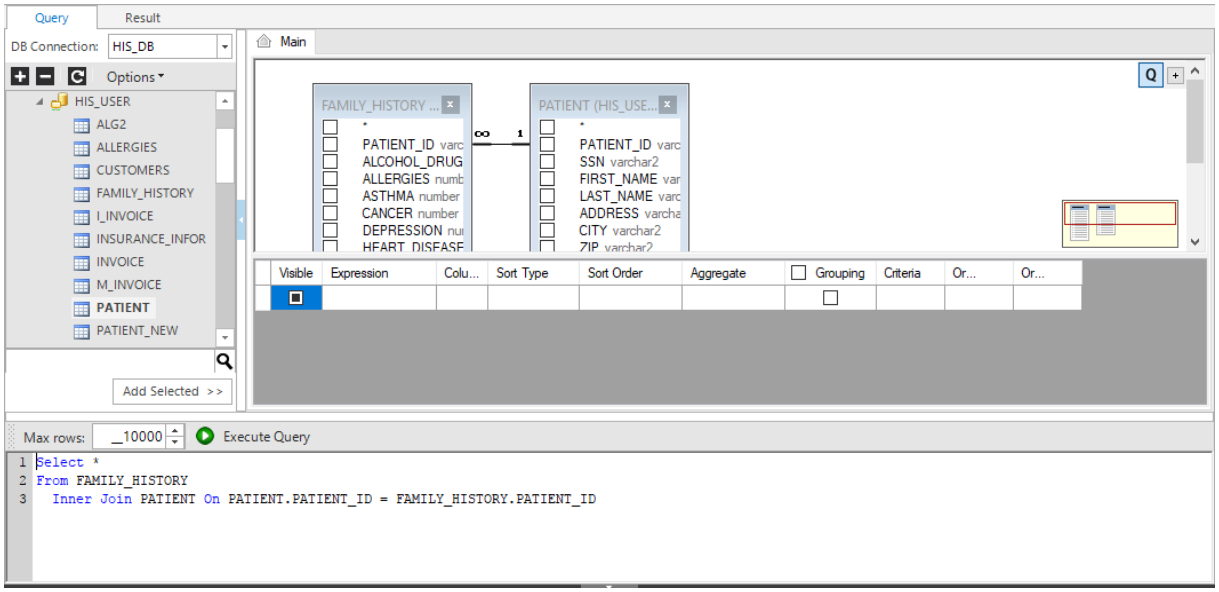
Displays the actual data that is received from executing the query.

17.2.2 Opening the Query Builder

To open the Query Builder window

(1) Click the **Query Builder** icon in Fabric left menu.

(2) The Query Builder window is displayed.



(3) Select the **DB Interface** to use from the **DB Connection** field.

The List of Tables in the DB is displayed in the DB Tree pane.

(4) Double click on the required DB table in the DB Tree or select the required DB table and then click on the **Add Selected** button.

The table is added to the main window, and its SQL statement is displayed in the query window.

(5) Edit the SQL query to add any Alias, Criteria, Sort, and Order either directly in the Query window or in the Main window.

(6) Click the **Execute Query** button.

Executes the SQL statement and displays the results.

> **Note:** DB objects can be refreshed by clicking the **Refresh** icon that appears next to the **DB Connection** dropdown list.

Options – Additional options to execute data loaded from databases:

- Include Synonyms

> **Note: In case of Fabric DB it is required to run Fabric commands on the left side of the window, such as set sync on; get LU.1; before running a select query on a table, as select must run on LUI level**

17.3 Auto Discovery Wizard

The Auto Discovery wizard enables automatic generation of a new Logical Unit DB based on predefined databases constraints (primary key, foreign key) in a few simple steps.

The Auto Discovery Wizard can run in two modes:

- **Fast** – the wizard generates a new Logical Unit schema based on foreign keys only.
- **Medium** - the wizard generates the new Logical Unit schema based on primary keys, foreign keys, and virtual primary keys.

17.3.1 Auto Discovery Wizard – Basic Step

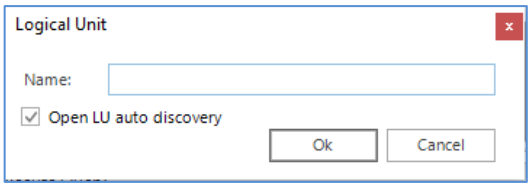
To create a new Logical Unit using the Wizard

(1) Right click the **Logical Units** in the Project Tree.

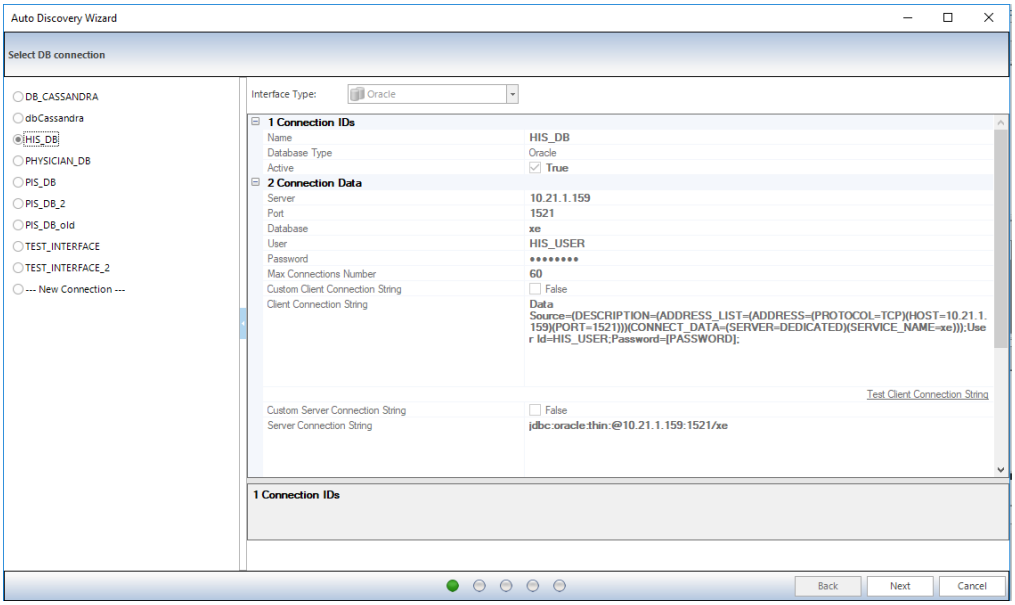
The Context menu is displayed.

(2) Select **New Logical Unit**

The Wizard window is displayed.



(3) Name the new Logical Unit and click **OK** to continue.



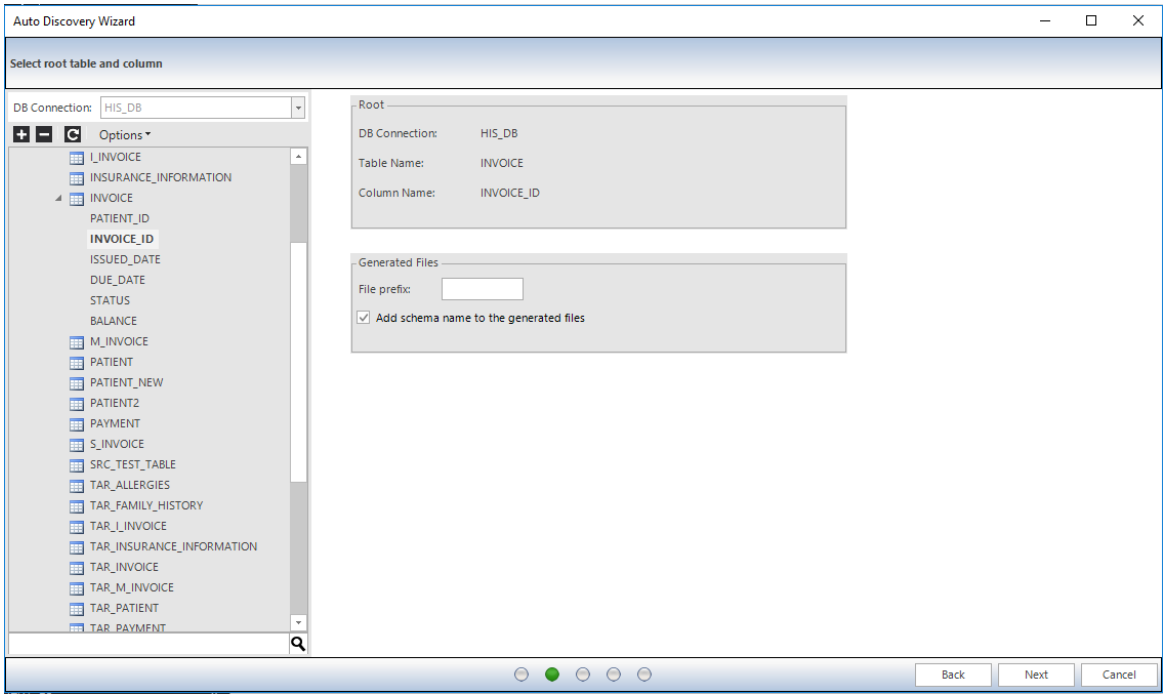
(4) The Interfaces selection window appears.

(5) Select the requested interface in the left pane. Once an interface is selected, the connection details are displayed in the left pane.

(6) Click **Next** to continue.

The root Table selection window appears.

In this window, select the preferred Table and Field, which define the root of the new Instance and its PK.



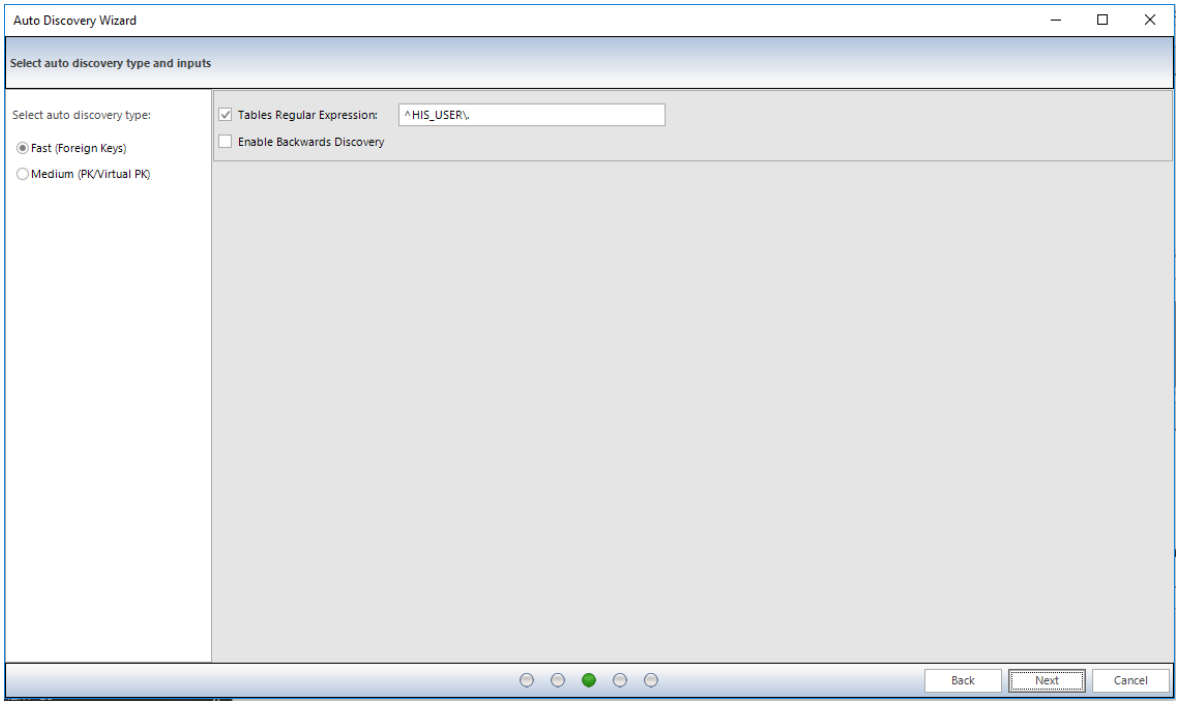
*The **Root** frame (upper right) is populated with information regarding the root table of the Logical Unit and the Instance ID for LU.*

(7) Enter a string prefix for the LU table names in the **File name prefix** field.

> **Note:** By default, the name for the new table is the name of the original source DB table.
>
> **Note:** Select the **Add schema name** checkbox, if adding the schema name is important.
>
> **Note:** This is only applicable for some databases (for example DB2, PostegreSql, SqlServer) and it is checked by default.

(8) Click **Next**. (0)

The second Auto Discovery Wizard window is displayed.



17.3.2 Fast Mode Discovery

In Fast Mode, the Auto Discovery Wizard follows the connections of the foreign keys. To limit the number of tables included in the auto discovery wizard, you can define a subset of the source DB tables to connect. The subset is defined by a regular expression that matches the names of the tables to include. For example, if you define the regular expression "**^CUSTOMER"**, the auto discovery wizard searches for relationships only within tables whose name begins with "CUSTOMER".

☞ To continue the Auto Discovery Wizard in Fast Mode

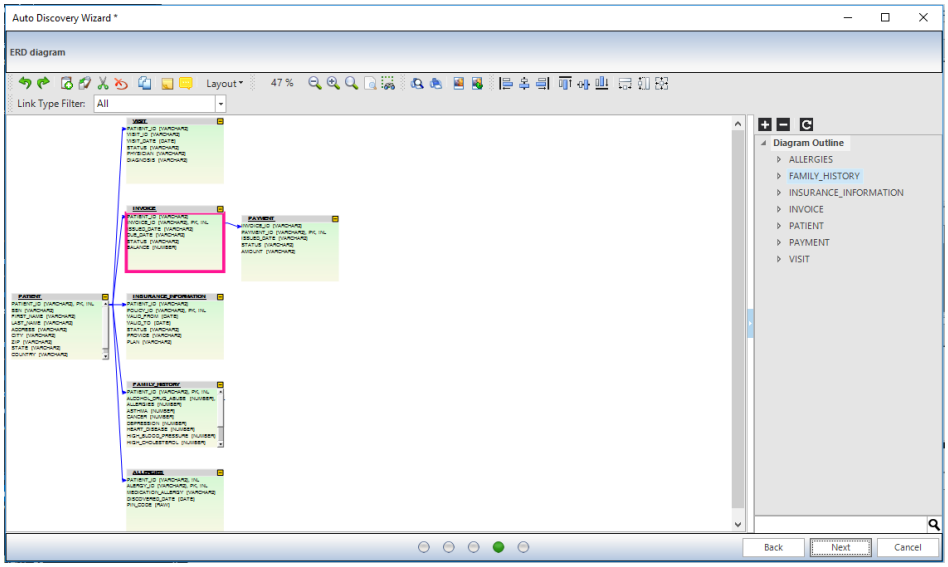
(1) Perform steps 1-8 of the basic procedure in Section 17.3.1.

(2) Click the **Fast** button in **Select auto discovery type** options (in the left panel of the window).

(3) Select the **Table Regular Expression** checkbox.

(4) Edit the regular expression in the field to the right of the **Table Regular Expression** checkbox.

(5) Click the **Next** button.

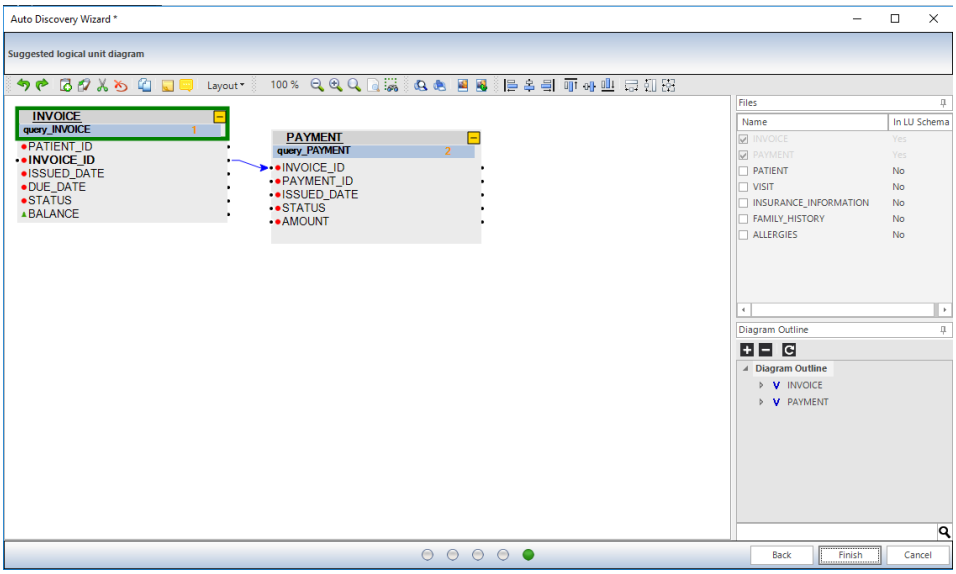


The Schema tables and their connections are displayed in the ERD diagram window.

> **Note:** The connections appear as blue lines to indicate that they represent foreign keys connections.

(6) Click **Next**.

The Suggested Logical Unit Diagram is displayed in view-only mode.



(7) Edit or delete tables from the Schema as described in Chapter 7.

(8) Click **Finish**. (0)

The new Logical Unit is displayed in the project tree.

17.3.3 Medium Mode Discovery

Medium mode allows you to be more involved in certain aspects of the process. In this mode you define:

- Column Aliases Collections - define the relationship between fields within different tables.
- Virtual Primary Keys - fields that were not originally defined as primary keys, but you want them recognized and treated as a primary key by the auto discovery wizard in addition to the originally defined primary keys.
- Primary Keys that are skipped - primary keys that are ignored or skipped by the auto discovery wizard.

☞ To continue the Auto Discovery Wizard in Medium Mode

(1) Perform steps 1-7 of the basic procedure in Section 17.3.1.

(2) Click the **Medium** button in **Select auto discovery type** options (in the left panel of the window).

The right side of the window is redrawn with a left pane displaying the DB tree of tables. The right side of the new area is split into three areas.

(3) Click the **Add** button in the **Set column aliases collection** area (top right).

The New empty line is added to the list of aliases.

(4) Drag column names (from different tables) from the DB tree (in the left pane) into the new line.

The Column name, qualified with a table name, is added to the newly added line.

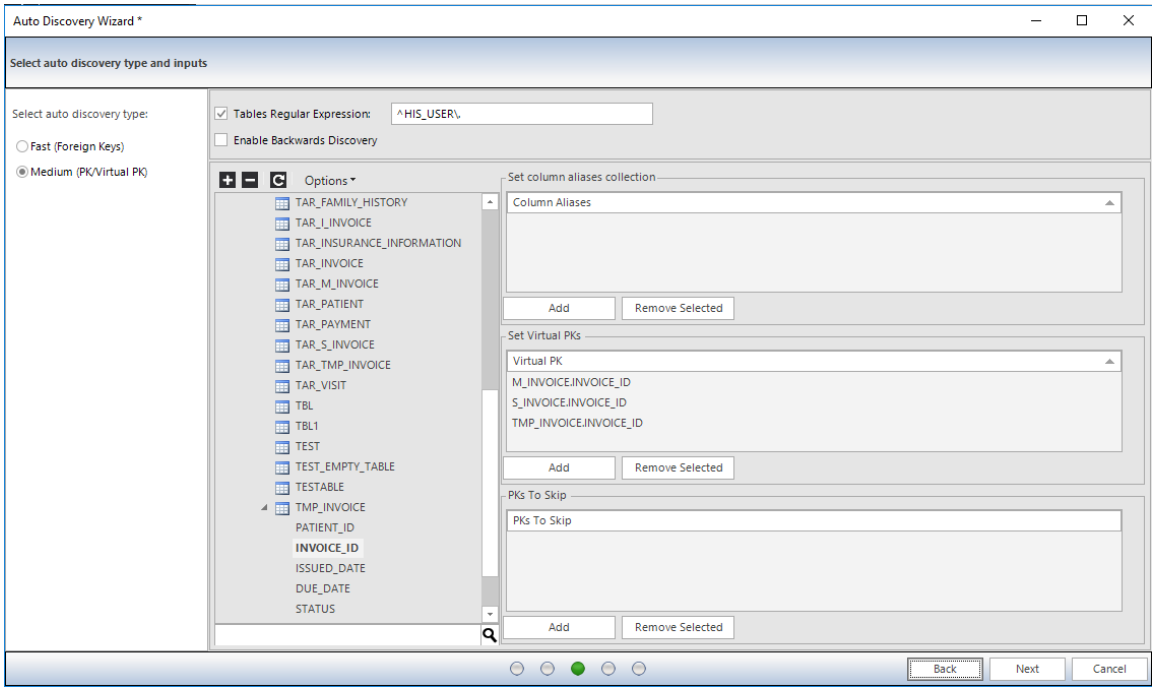
> **Note:** The columns are listed as <Table Name>.<Column Name>.

(5) Click the **Add** button in the **Set Virtual PKs** area (middle right).

The New empty line is added to the list of virtual primary keys.

(6) Drag column names (from different tables) from the DB tree (in the left pane) into the new line.

The Column name, qualified with a table name, is added to the newly added line.



(7) Click the **Add** button in **PKs To Skip** (bottom right).

The New empty line is added to the list of primary keys to skip.

(8) Drag column names (from different tables) from the DB tree (in the left pane) into the new line.

The Column name, qualified with a table name, is added to the newly added line.

(9) Select the **Table Regular Expression** checkbox.

(10) Edit the regular expression in the field to right of the **Table Regular Expression** checkbox.

(11) Click the **Next** button.

The tables and their connections are displayed in the ERD diagram window.

> **Note:** The connections line colors indicate the type of connection they represent -

- Primary key connections in green.
- Virtual foreign key connections in olive.
- Foreign keys connections in blue.

(12) Click **Next**.

The Suggested Logical Unit Diagram is displayed in a view-only mode.

(13) Click **Next**.

The Suggested Logical Unit diagram is displayed.

(14) Edit or delete tables from the Schema as described in step (5) above.

(15) Click **Finish**. (0)

The new Logical Unit is displayed in the project tree.

17.3.4 Adding Auto Discovery Functionality to an Existing Schema

This feature can be useful when there is a need to add additional tables to an existing Schema from a different interface.

☞ **To use Auto Discovery to add tables to a Schema**

(1) Select a column within the Logical Unit DB from the project tree.

(2) Right click the column name.

The Context menu is displayed.

(3) Select the **Auto Discovery Wizard** option.

(4) Continue as described in Section 17.3.2 or Section 17.3.3.

18 Customer Support

Please contact our offices for further information on K2View products:

K2View email: support@k2view.com

19 Graphit Utility

19.1 Graphit Overview - Generating Dynamic CSV/XML/JSON Documents

Graphit is a Fabric utility that enables the user to create dynamic CSV, XML and JSON documents.

The Graphit utility is very useful for the generation of Fabric web services' responses. The response content will be defined at execution time, according to the specific parameters relevant to each web service call and the logical unit instance that is used, or by retrieving dynamically information from other databases or interfaces.

19.2 Graphit Main Features

The main features of the Graphit utility include:

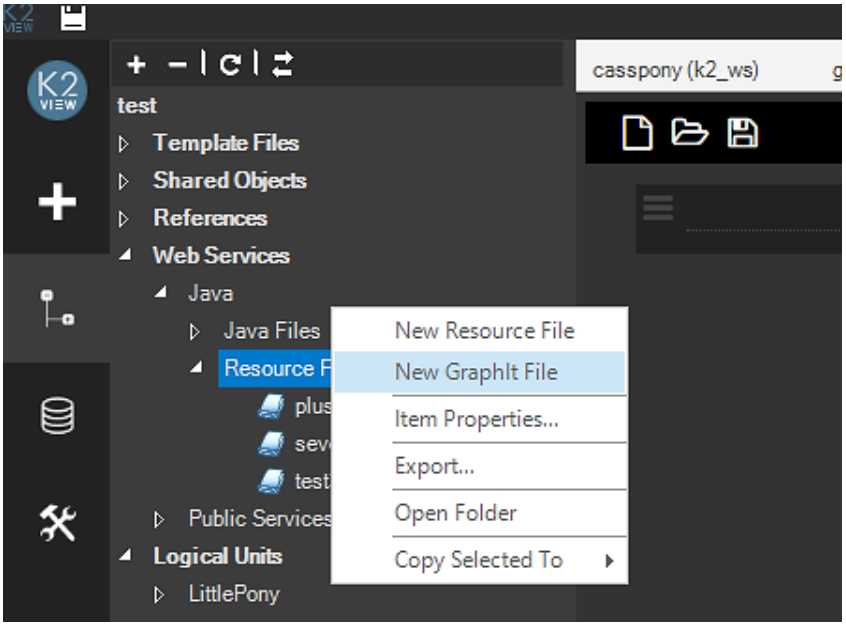
- Supporting CSV, XML and JSON documents creation.
- Executing a query on the LUDB, iterating over the results and using the returned values in the creation of the response document.
- Accepting external input as variables.
- Using variables in queries.
- Executing the Queries on any interface (not just Fabric).
- Supporting prepared statements.
- Supporting the hierarchical nature of XML and JSON, including the ability for a query in one of the inner levels of the hierarchy. Use any "outer level" query result as an argument.
- Recursively generating nested tags and structures.
- Allowing to customize the values formats as well as to define whether to generate a tag when the values are null or empty.

19.2.1 Creating a New Graphit File

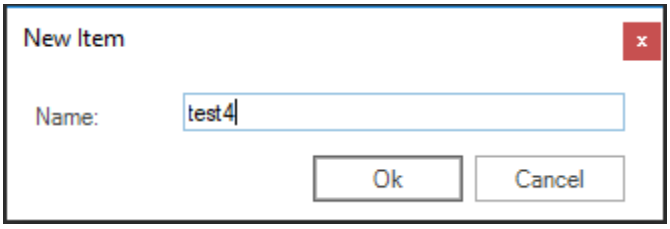
(1) Open the K2View Fabric Studio software.

(2) Expand: Web Services >> Java >> Resource Files.

(3) Right-click Resource Files and select New Graphit File.

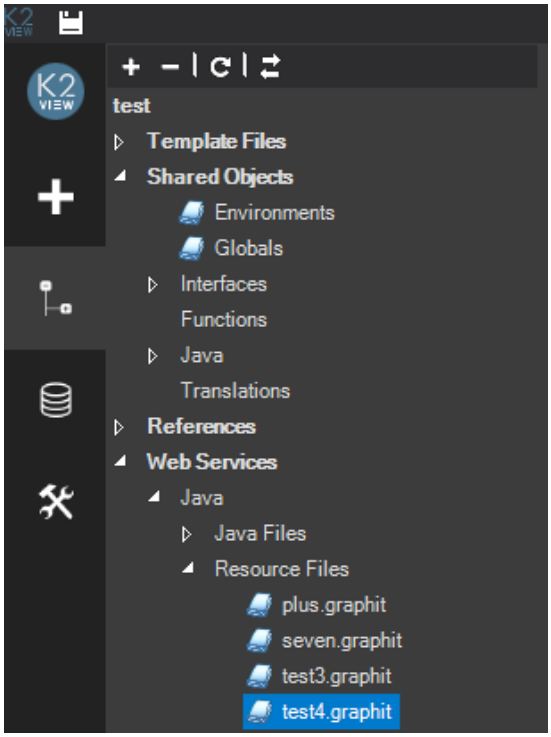


(4) Assign a name to the new Graphit file and click OK.



Note: The file name must start with a letter and contain alpha-numeric characters

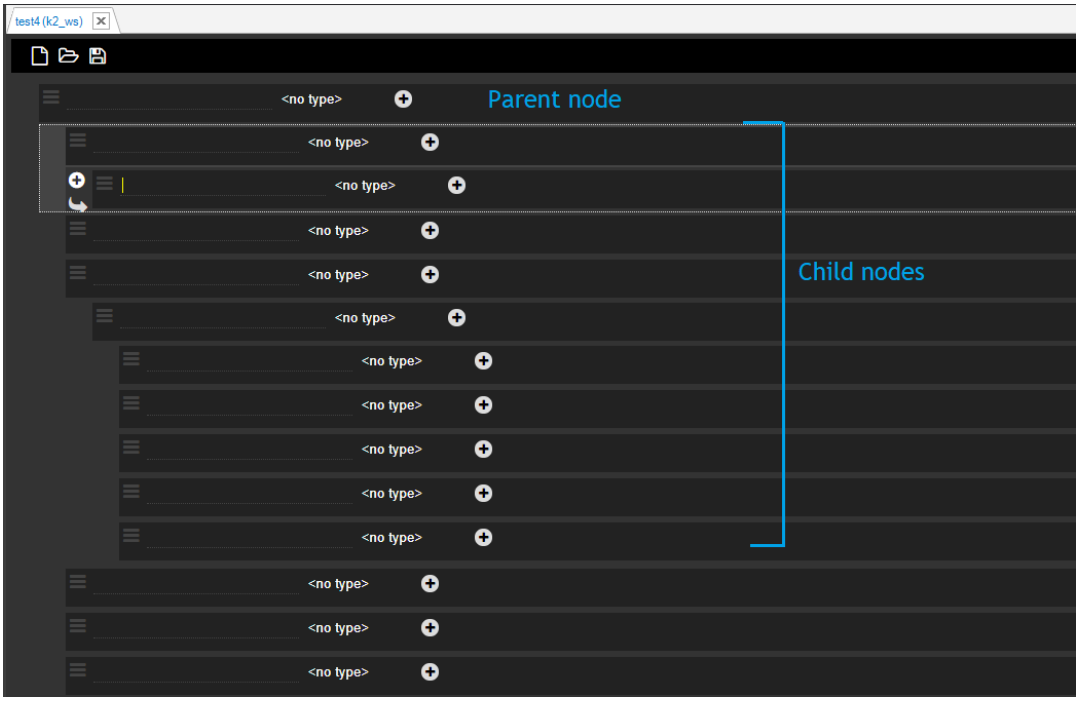
The file is created and displayed on the project tree under the Resource Files.



19.2.2 Editing a Graphit File

Once a new Graphit file iscreated, the user can edit it in order to create the required CSV/XML/JSON document structure. The Graphit file is presented as a hierarchical structure of nodes, where each node defines a tag or condition in the structure of the resulted CSV, XML or JSON document. Each of the nodes can have child nodes and the child nodes can also have child nodes, thus creating nested tags in the resulted document.

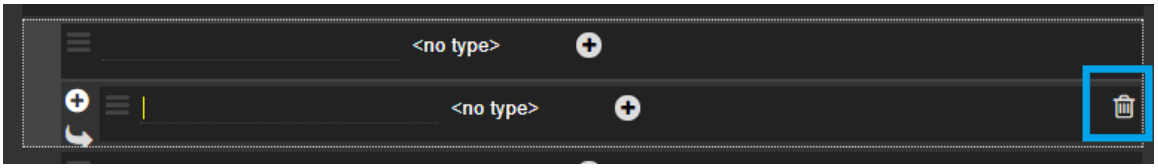
For each of the nodes, node Name, Type and properties can be defined in order to instruct how to generate the result. The different options are detailed in the next sections.



- To create a parent node, click the 'plus' button. A new node at the same level is created under the original node.

- To create a child node, click the arrow button. A new child node is created under the parent node.

- To delete a node, click the bin icon next to the node. The node is deleted.



To change the node location inside the hierarchy, click and drag the desired node to another location.

19.2.3 Assigning Name to a Node

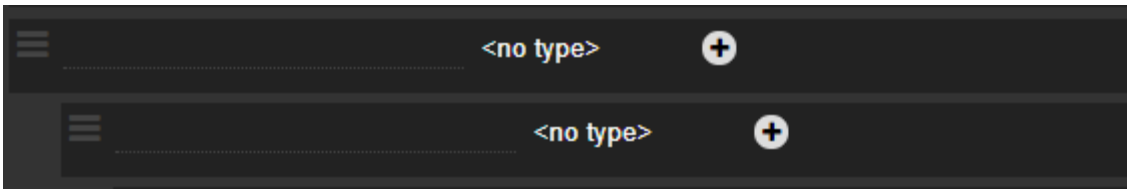
Every node must be assigned a name in order to appear in the output document.

If no name is assigned, the node will not appear in the output. Node without name may be used for internal purposes.

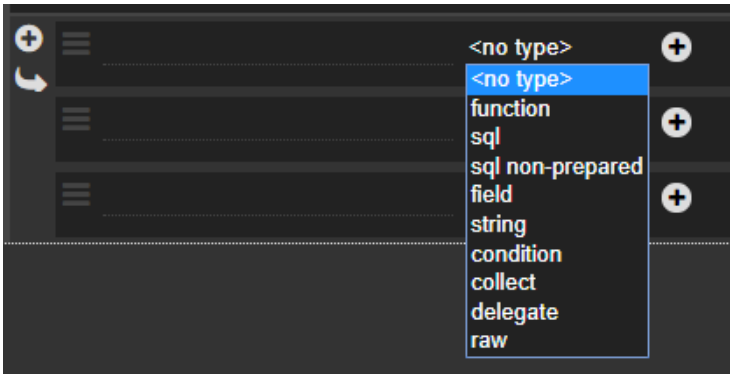
To assign name to a node, set your mouse in the left side of the frame created for the node, and write the node name. This name will be the tag name, if this node is presented in the output document.

19.2.4 Assigning Type to a Node

The Node type controls different options for the presentation and content of the respective tag in the output document. By default, the created nodes are not assigned any type or property.



To assign a type for the node, click the type field to the right side of the node name to open a dropdown list of types.



The available types are listed in the table below:

Type	Description
function	This node type is used to run a code in order to determine the value of the node. The code should be written in java script.
sql	This node type provides a way to write an SQL, retrieving information from either the Fabric or from any other database interface. If the interface is other than Fabric, the interface name should be specified as a node property, as described in the Properties section. At run time, the SQL will be executed on the fabric and the results can be used in the nested nodes, as well as allow looping on all the results and executing nested nodes for each of the returned rows. SQL does not support * in the select statement. The table fields should be specified. The node of SQL type is used for internal processing when the document is prepared. In order for the returned values to be part of the result document, additional nodes of type Field should be added, where the selected response will be added as a tag in the response.
sql non-prepared	This node type is similar to the previous SQL type, except for query parameters that are not binded in advance.
field	This is the simplest type of a node – it turns the node to a tag in the XML/JSON
String	This node type should be used when there is a need to concatenate two or more values. The concatenation is done using the + sign.
condition	The node will include a condition, and according to the condition result, nested nodes will be executed or not.
collect	run a loop over collection of data
delegate	This node type should be used as creating parenthesis. All the nested nodes under it are included within the parenthesis.
raw	present the data as output with no manipulation. For example header for xml format

19.2.5 Assigning Property to a Node

Properties are additional instructions that can be given for each node, for example, how to format a number, what database to query, or if the node is active or disabled.

Click the + sign to the right of the node in order to open a list of available properties for a node and select the properties that should be assigned to this node.

The following options are available:

To add a property to a node, select the required property from the list and click "Add".

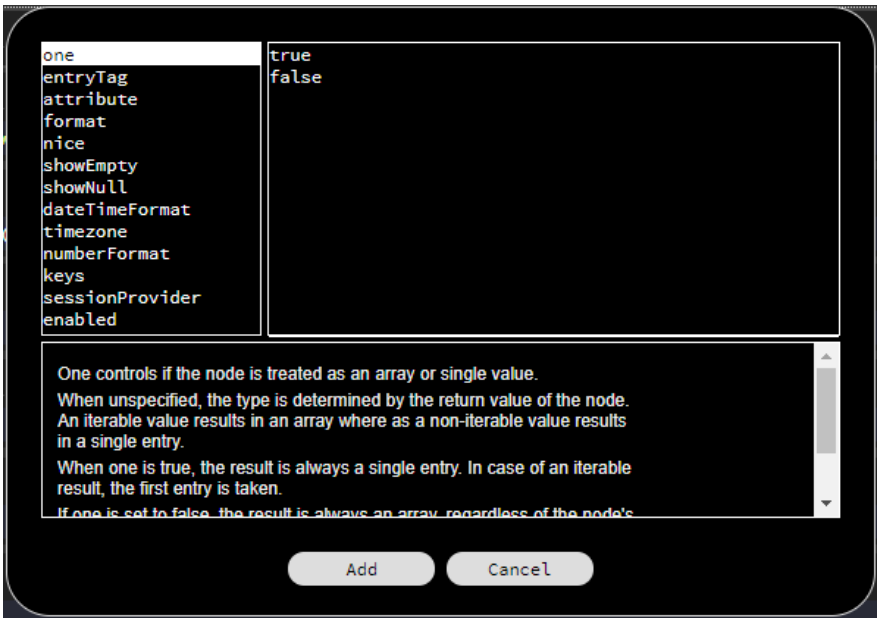
Property	Description
one	Select to define if the node is treated as an array or single value. When unspecified, the type is determined by the return value of the node. If one is true, the result is always a single entry, even if the query that it is based on returns multiple rows. When false, the result is always an array. Valid values are true or false. For more information, see the instructions displayed for this property in the property selection window.
entry tag	Select to define the tag that surrounds XML array entries. If 'none' is specified, the value 'entry' is used. This property affects only the node it is set on.

Property	Description
attribute	Select to determine if, in the case of XML, the value is set as an attribute or child node. The default is child node. This property only affects the node it is set on. For more information, see the instructions displayed for this property in the property selection window.
format	Select to define if the node will only be evaluated or presented if the output format matches the format value. For example, a tag that should only be printed if the format is XML. This property affects only the node it is set on. The valid values are XML or JSON.
nice	Select to define if the output format will be formatted in a "nice" layout, meaning every tag printed in a new line and indentation to facilitate reading (set to true). The default is false. This property affects the node and all its child nodes.
showEmpty	Select to define if empty arrays are shown in the output. The default is true. This property affects the node and all its child nodes.
showNull	Select to define if null entries are shown in the output. The default is true. This property affects the node and all its child nodes.
numberFormat	Select to define how numbers will be formatted in the output. You can use one of the built in formats or define your own format, using the syntax that is described in the property window when selecting the numberFormat option.
keys	Select to replace nested queries by joining the data on the root query and grouping it by key. In the scope of a query, keys is used to select a subset of rows to group for each invocation of the node. When keys is specified in child nodes, each node will group the tows of it's parent nodes, according to the key. For more information, see the instructions displayed for this property in the property selection window.

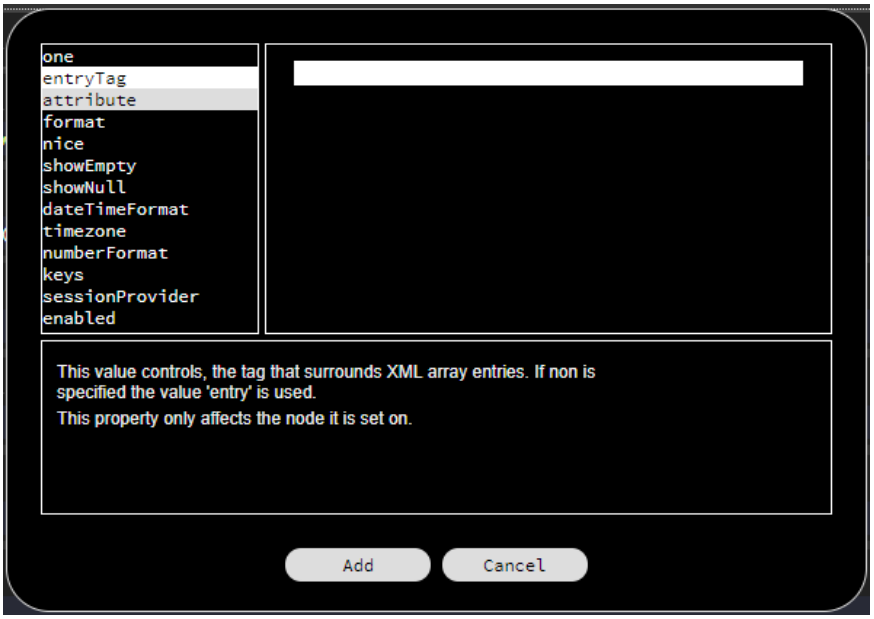
Property	Description
sessionProvider	Select to define which interface should be used for a query. This property should be defined each time a node is defined as “SQL” or “sql non-prepared” and the database to be queried is not the fabric itself. This property affects the node and all its child nodes.
enabled	Select to define whether the node and its children will be evaluated. If set to false, the node and its children will not be evaluated.
csvHeader	Disable or enable the CSV header row. By default the CSV format display the header row.
csvRow	Define the delimiter between rows in the CSV format. The default is the the CR (\n) sign. The format supports \n \r \t for CR LF and Tab.
csvCol	Define the delimiter between columns in the CSV format. The default is the the comma sign. The format supports \n \r \t for CR LF and Tab.
csvEnclose	Define the character used to enclose a value in the CSV format. This is used only if the value contains a special character (csvEnclose, csvRow, csvCol). In case the value contains the csvEnclose character, it is doubled. For instance, using the default values, the string ab"c will be serialized as "ab""c" The format supports \n \r \t for CR LF and Tab.

The different attributes and their values are presented in the images below.

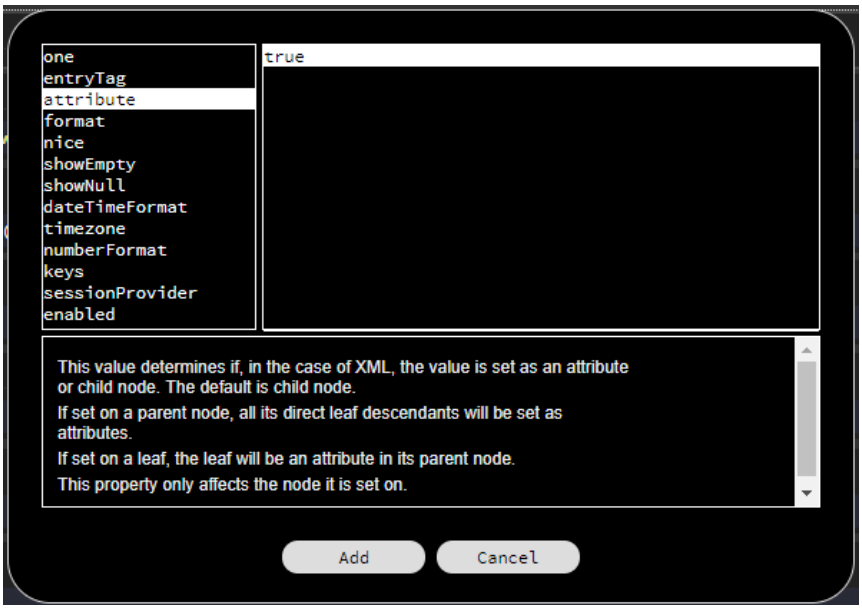
One



entryTag

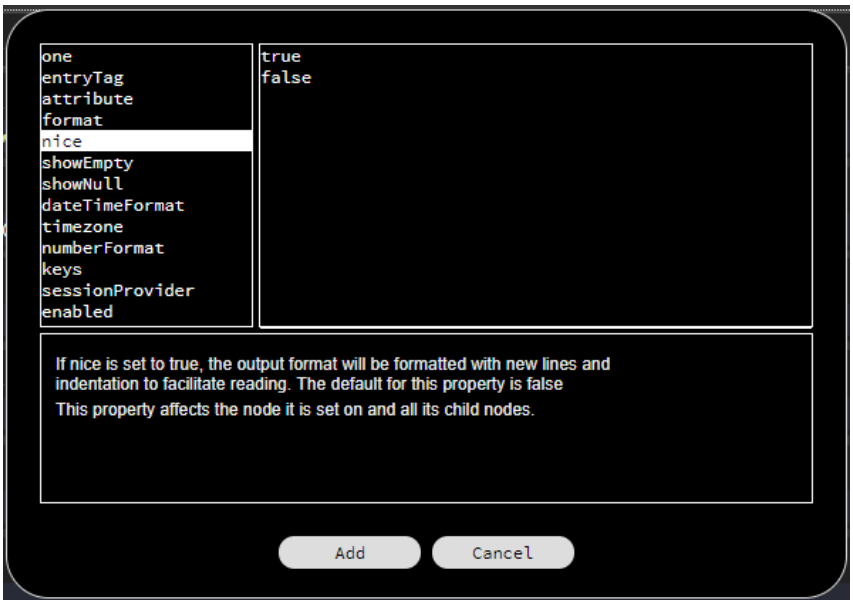


attribute

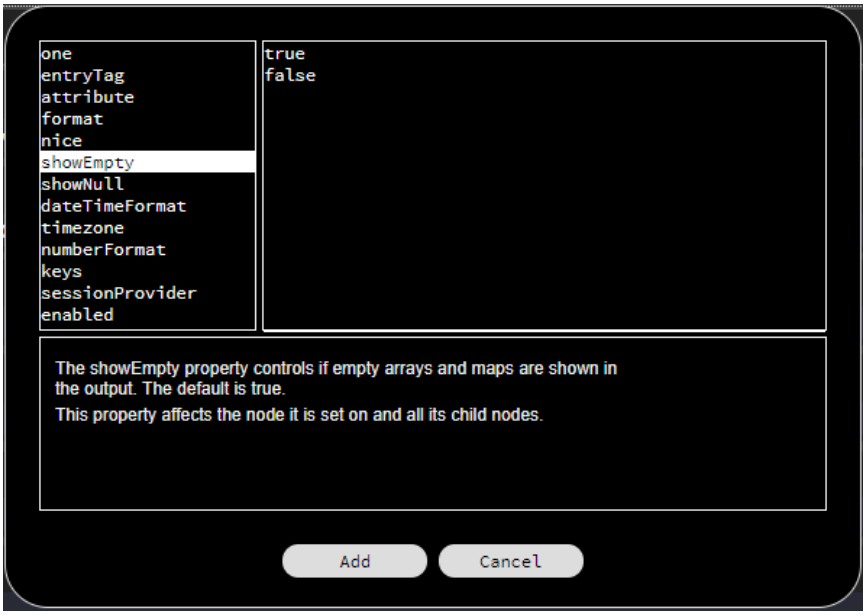


format

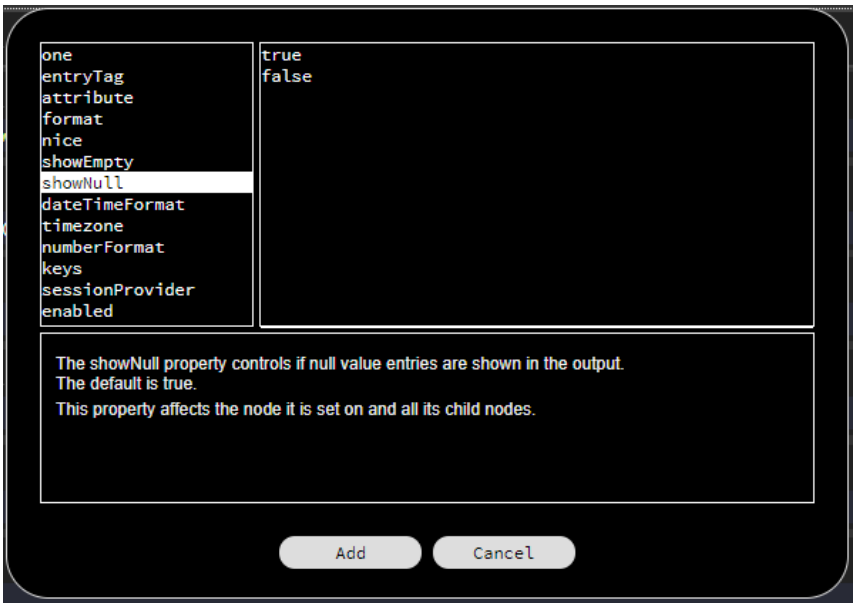
Nice



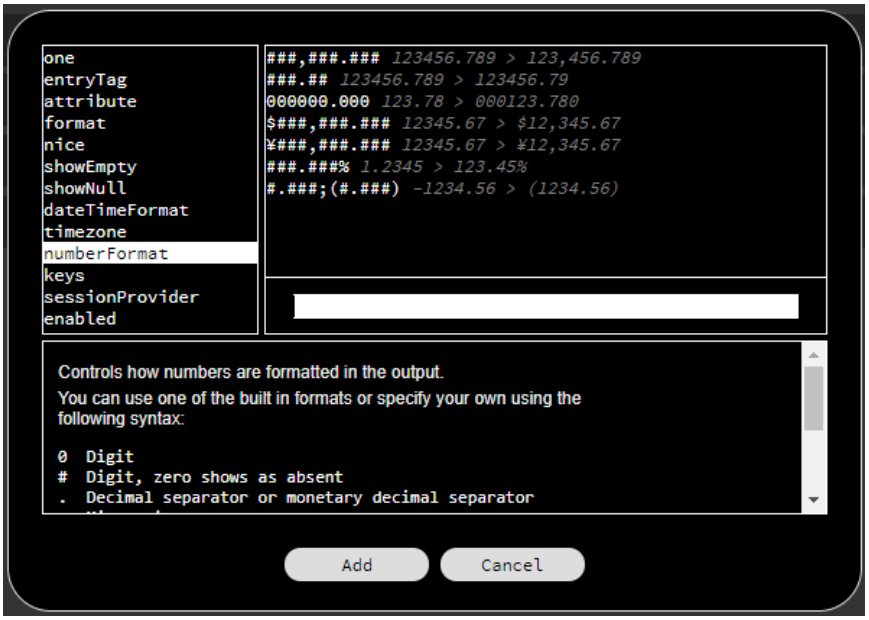
showEmpty



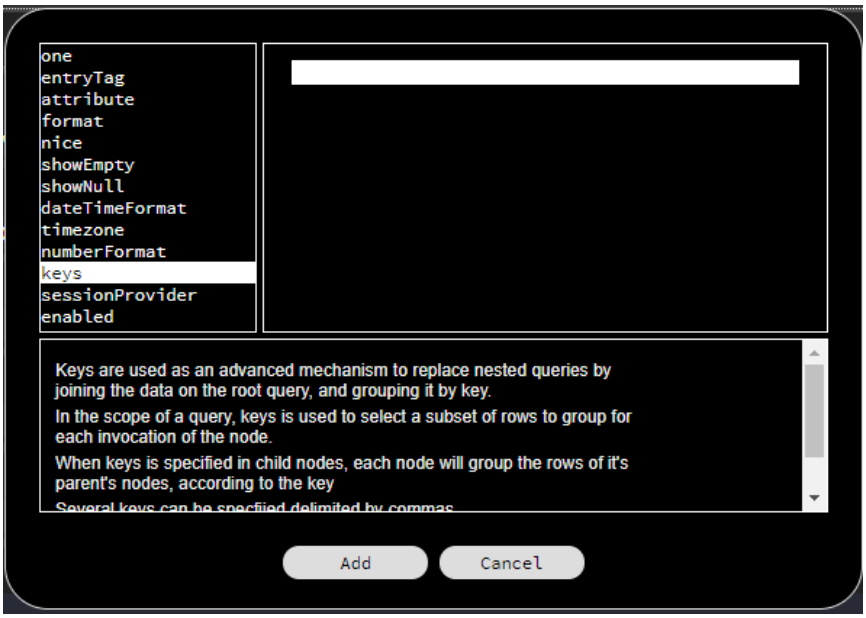
showNull



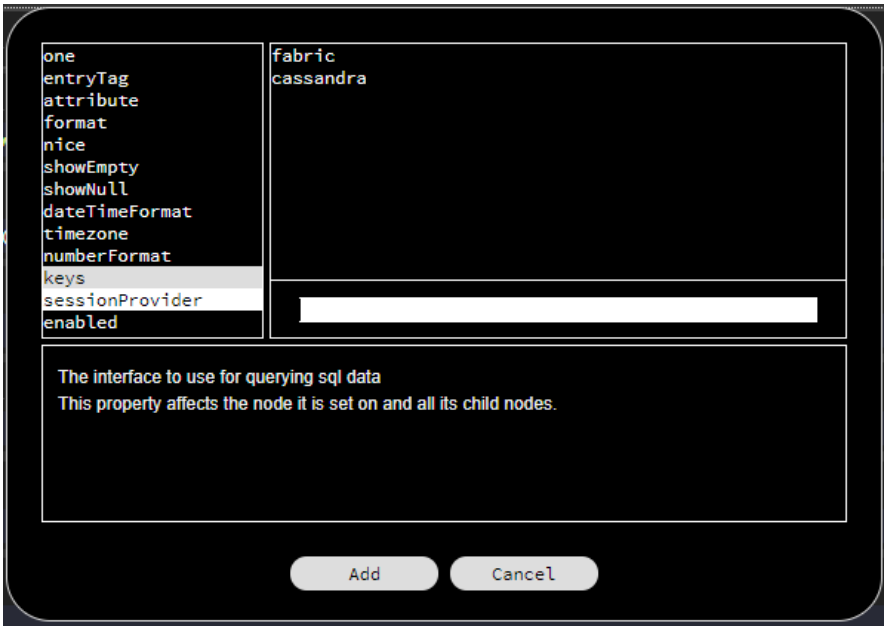
numberFormat



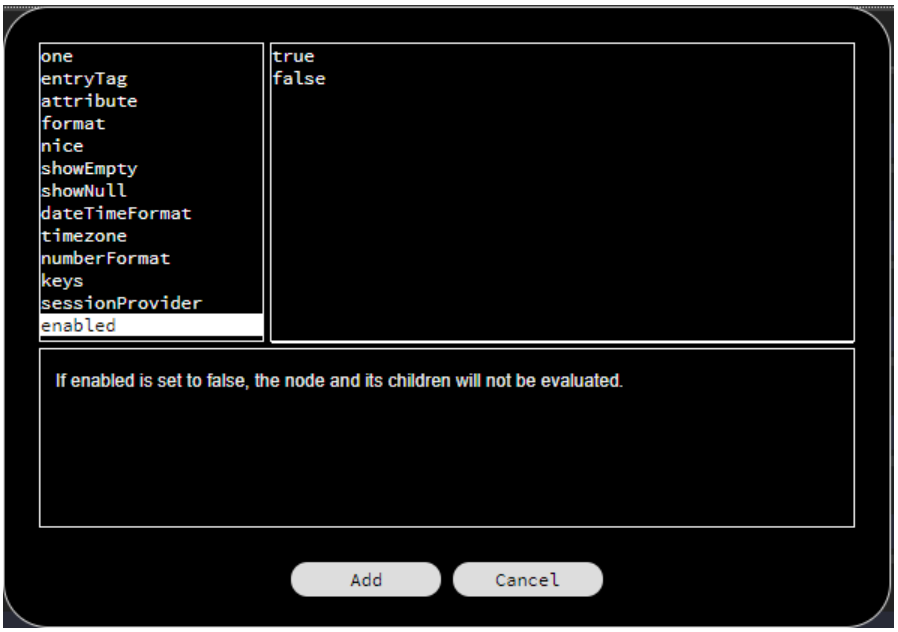
keys



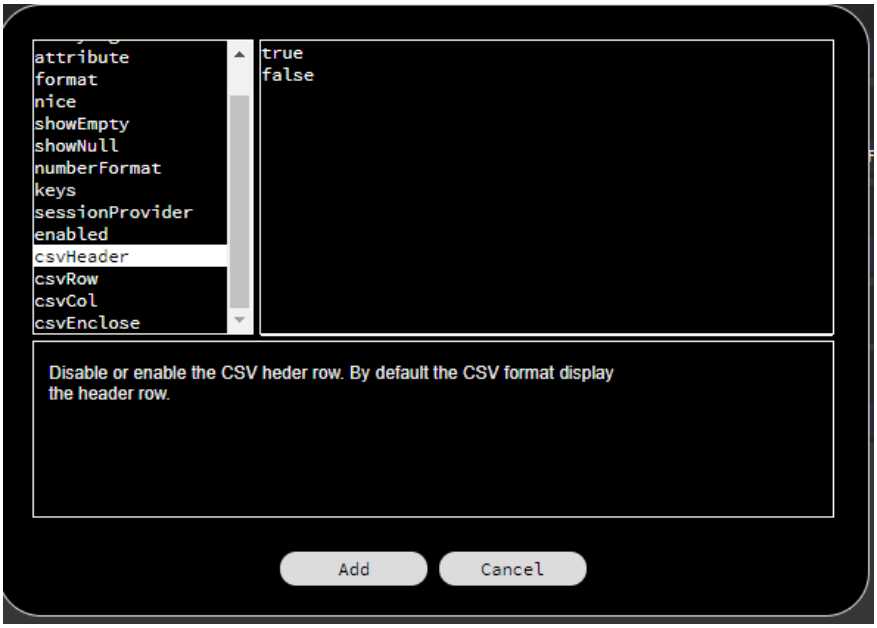
sessionProvider



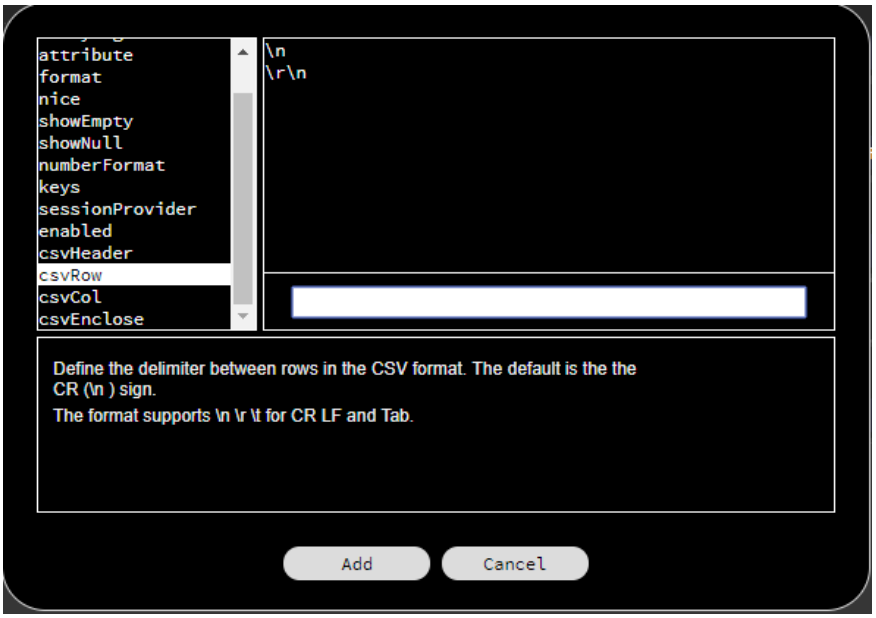
enabled



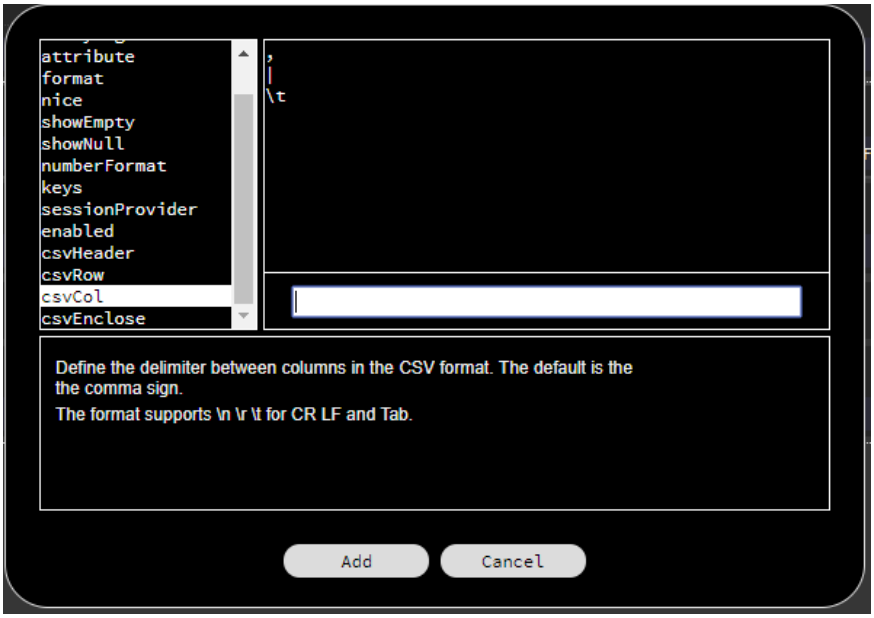
csvHeader



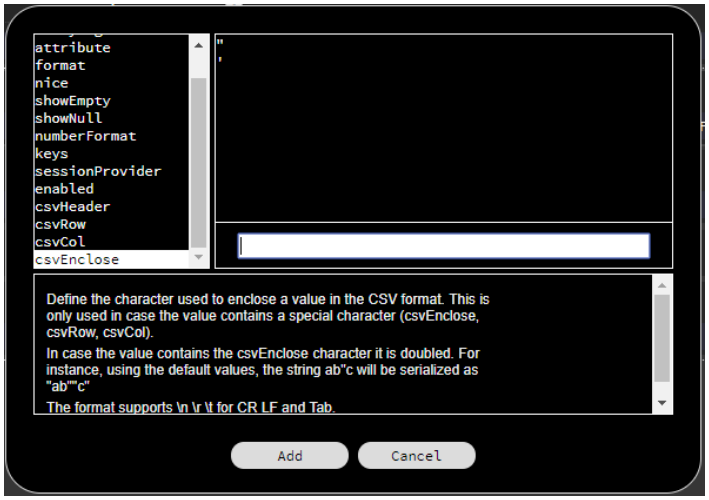
csvRow



csvCol



csvEnclose



19.2.6 Debugging a Graphit Document

Graphit is provided with a debug option that allows testing the resulted XML or JSON without having to deploy the implementation.

The debug can be done using an LUDB file that was generated previously in the Fabric LUDB viewer, or it can be executed by pointing to the Fabric on the server that is defined for deploying the project.

In both cases, the implementer can set the values of any input parameter by clicking the Parameters option on the top left side of the Graphit window.

The Profiler option is used in order to profile the performance of the Graphit code and to provide running time for each one of the Graphit sections.

19.2.7 Using Graphit File as Part of a Web Service

The most frequent use of the Graphit file is within a web service, with the purpose of constructing the web service response.

In order to use the Graphit file, include the following code in your web service implementation:

New Object response = graphit(<file name> , <Input parameters>)

In the code above, the “response” variable will get the CSV, JSON or XML response string, and can then be returned as the web service output.

The function parameters are:

- File_name = the name you assign the Graphit file that should generate the response document. If the web service name is the same as the Graphit file name, this parameter can be set to null.
- Input parameters – comma separated list of the parameters expected by Graphit file, for example, the logical unit instance key.

19.2.8 Invoke a Java Code from Graphit

You can bind java functions to the Graphit scope using a web service. You need to create a Lambada expression based on the Scripter.F functional interface which has the following signature:

- Object f(Object... var1);

Limitation:

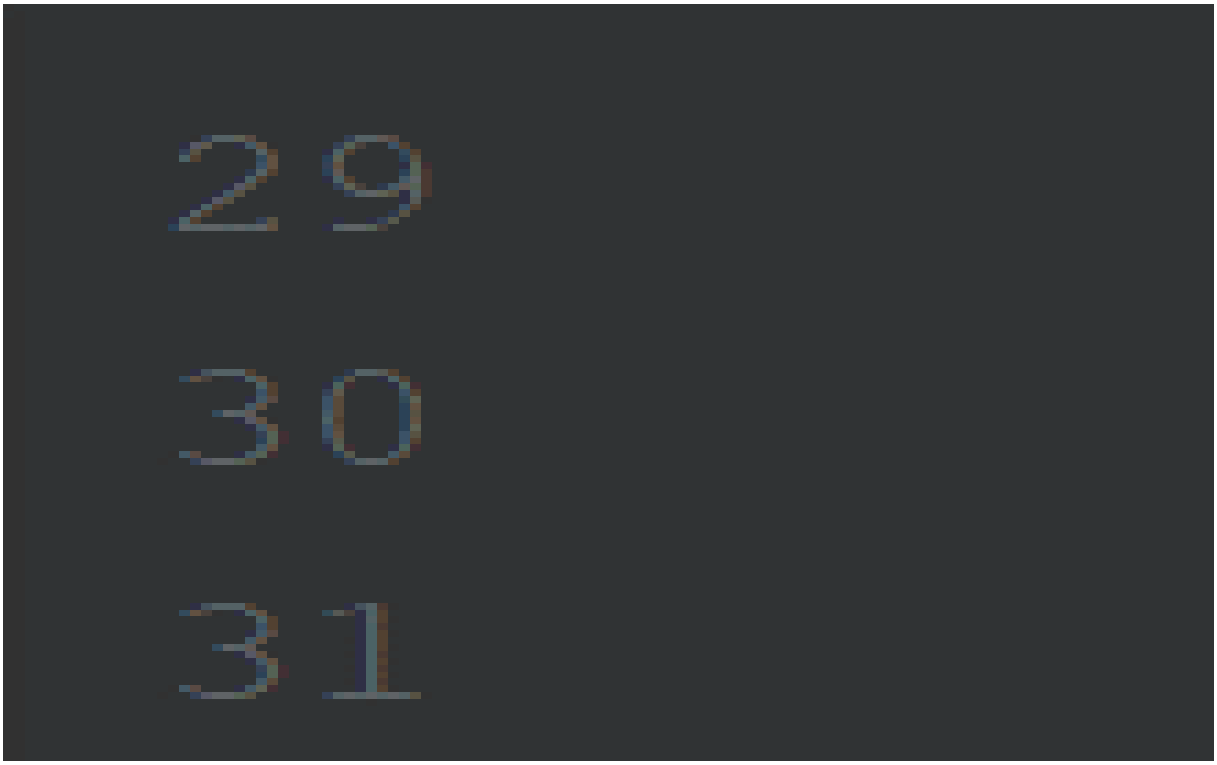
- This cannot be debugged in the Graphit editor (as the functional parameters cannot be created in the studio). You can test your code by deploying to Fabric or debugging the web service code using the IntelliJ editor.

19.2.8.1 Invoke a Java Code from Graphit-Example

The following Graphit code calls the following Java methods:

- someLogic
- times3
-

javaf_demo is the webservice code, demonstrating how to bind the functions to the Graphit scope.

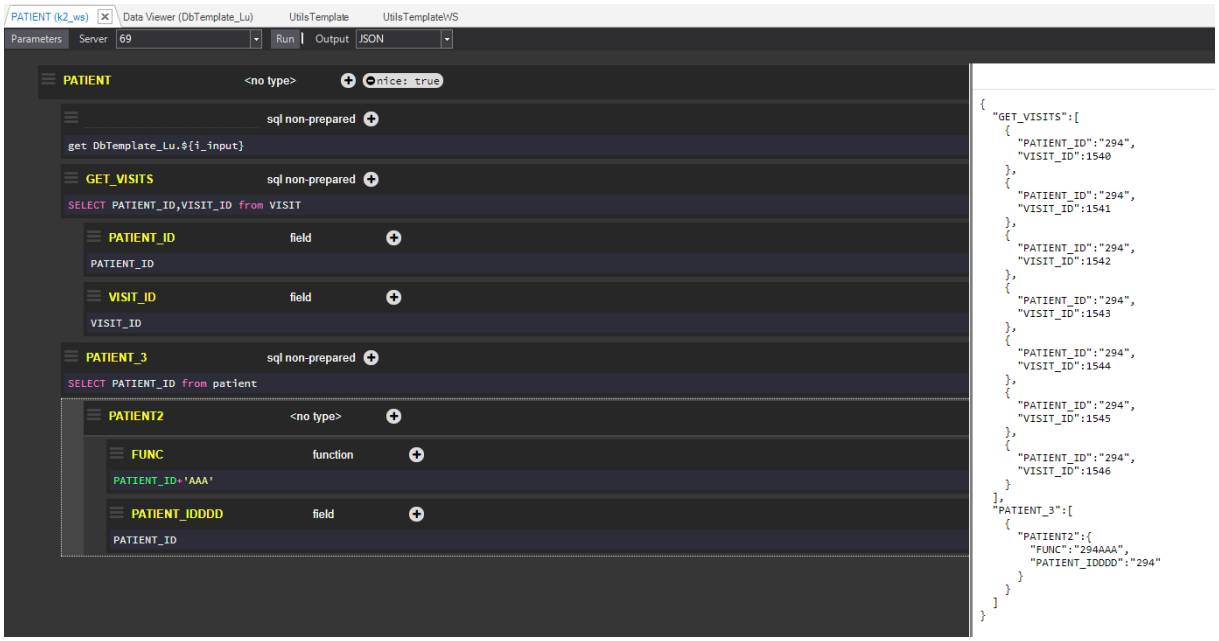


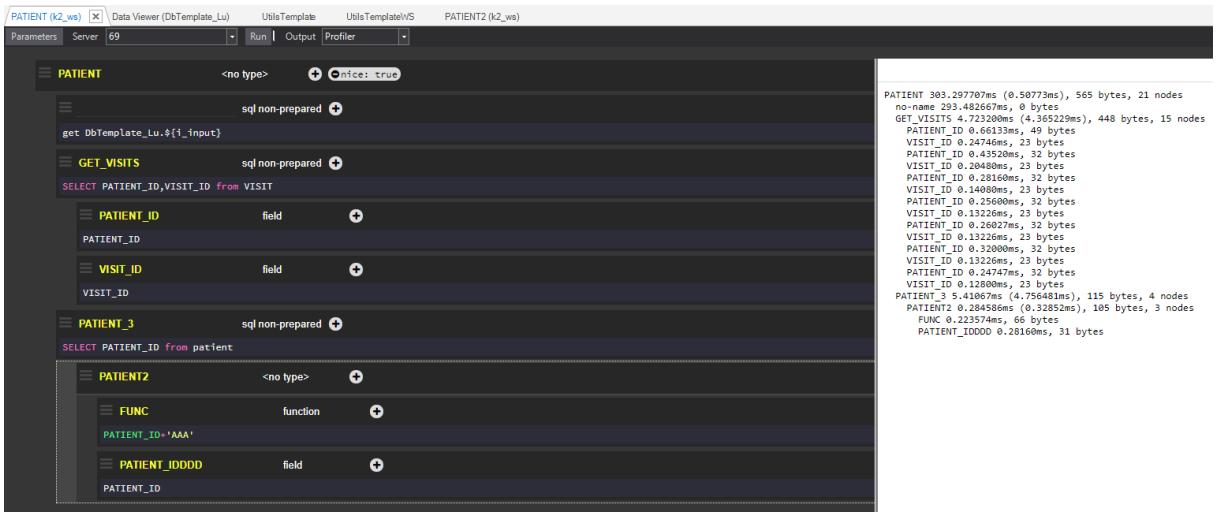
The result:

Response Body


```
[
  {
    "field1": "a string: 13.5 a map",
    "field2": 13.5
  }
]
```

19.2.9 Examples





20 Integration with IntelliJ

20.1 Overview

Fabric Studio was enhanced to fully integrate with IntelliJ, meaning that the Java Code (Functions/Globals) can be developed on Studio or IntelliJ and a full sync between the two tools is done automatically.

All IntelliJ capabilities can be used in Fabric Studio (automatic completed by context Java/Product functions, etc).

Full debugging capabilities were enabled supporting the following debugging levels:

- LU
- Population
- Web-Service
- Parser

When right clicking the function, the following option was added (after the function is saved):

Open in IntelliJ option opens the function on IntelliJ exactly in the same line that was pointed out in the studio.

20.2 Project Structure

Eliminates the XML files used for functions/globals in the project, using Java files directly.

When opening an existing project from release 4.0 and above, all functions and globals XML files are removed and Java files are created instead.

New files were created for the following levels (replacing the Utils.java file):

- Shared Objects
  - SharedLogic.java - default
    - contains all shared functions from all categories (except product functions).
  - SharedLogic.java – per category
    - Contains all shared functions for a defined physical category, each SharedLogic.java file will be placed under the relevant Category folder
  - SharedGlobals.java
    - contains all shared globals
  - Product functions are now part of server side, available only for dragging into population/parser maps

Functions/globals layout in projects tree moved to be under java logic/global

- LU/Reference/Web-Service Level
  - Logic.java
    - May have more than one Logic.java file, one per each category
  - Globals.java
    - contains all on LU/Reference globals

Export/Import is done on the Category level. If only one function from a category should be copied to the other category or project, you need to copy and paste the required function.

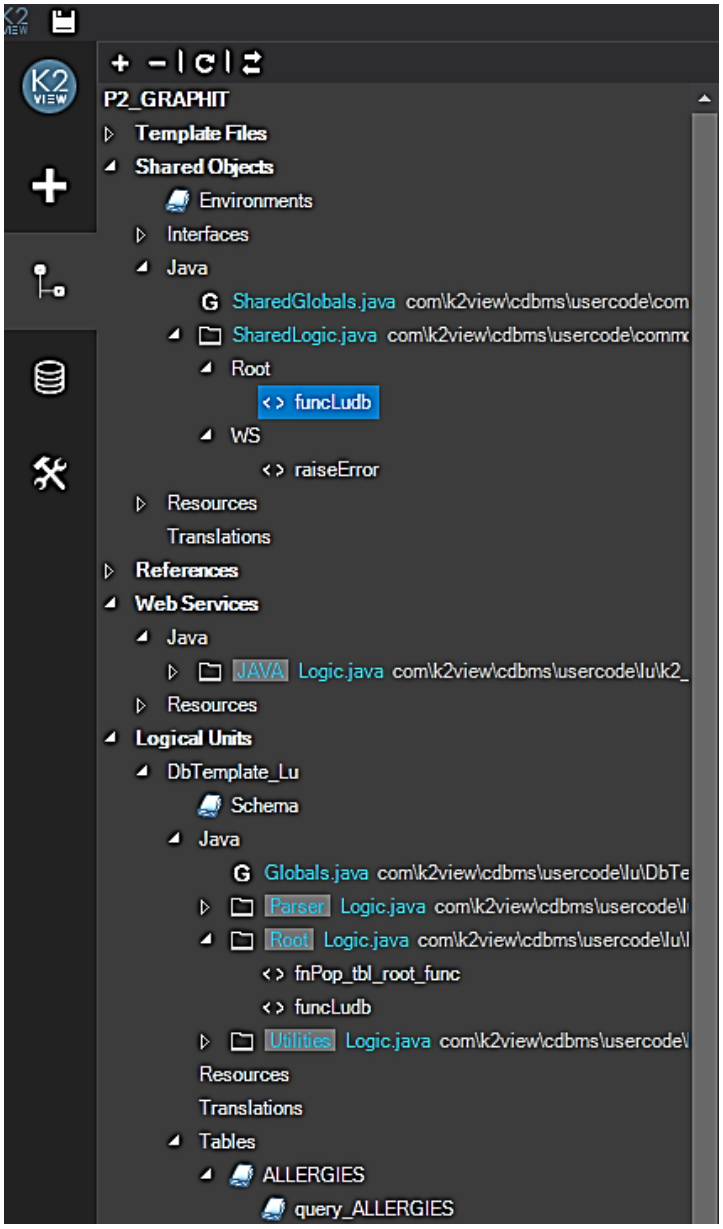
Version Control is maintained on the Globals/Logic.java level and not on the function level.

Link to IntelliJ download:
https://www.jetbrains.com/idea/download/#section=windows

> Note: After the installation, a JDK location needs to be defined.

Switching between the Studio and IntelliJ is done by pressing Ctrl+I (available only on functions).

Below is an example of a new tree structure:



20.3 Debugging

It is possible to debug in IntelliJ against a real running local fabric. It is a must to deploy the project first.

DEBUGGING is supported on the following levels:

- DEBUG LU(same as data viewer)
- DEBUG POPULATION(same as POPULATION debug in studio)
- DEBUG PARSER(same as PARSER debug in studio)
- DEBUG WEBSERVICE
- DEBUG BROADWAY flow
- DEBUG User Job

For all debugging level the following steps should be followed:

- In IntelliJ choose run->attach to process
- Choose the fabric debug process
- In IntelliJ Set a break point in the relevant function
- Run the fabric command

21 Appendix A: Built-In Functions

21.1 Overview

Built-in functions are part of the standard installation and enable you to access the internal database as well as external databases. These functions can be used within the Java code editor window only and are not visible on the maps.

21.2 Server Functions

21.2.1 fetch

Function: fetch	
Format	Db.Rows fetch(String sql, Object... params) Db.Rows fetch(Db.Statement statement, Object... params)
Description	Return the rows based on an SQL String and parameters.
Input	Sql – The select SQL to be executed params – Values used for the prepare statement
Output	Db/Rows object that holds the returned results.

Function: fetch

Examples

Run select on the ludb for a specific instance with and without parameters:

```
Db.Rows rows = ludb("PATIENT_LU", 1).fetch(sql, “XX”, 123);

Db.Rows rows = ludb("PATIENT_LU", 1).fetch(sql);
```

Run select on interface-

```
String selectRefTablesStats = "Select count(*),
to_char(min(start_time), 'YYYY-MM-DD HH24:MI:SS'),
to_char(max(end_time), 'YYYY-MM-DD HH24:MI:SS'), execution_status
from TASK_REF_EXE_STATS where task_execution_id = ? group by
execution_status";


Db.Rows rows = db("TDM").fetch(selectRefTablesStats,
refTaskExecutionId);
```

Run select on Fabric –

```
String sql = "SELECT PATIENT_ID, INVOICE_ID, ISSUED_DATE, DUE_DATE
FROM PATIENT_LU.INVOICE where invoice_id > ? ";

Db.Rows rows = db("fabric").fetch(sql, invoiceId);

// Option 2

Db.Rows rows = fabric().fetch(sql, invoiceId);


// Note- you need to use question mark when using the binding
parameters in the SQL query


Loop on the returned results-

for (Db.Row row:rows){

  String patientId = row.cell(0).toString();

  String invoiceId = row.cell(1).toString();

}


Get the Instance in the Scope for Customer LU

String IID= db(“fabric”).fetch("select
IID(‘Customer’)").firstValue().toString();


Get the version of the Instance in the Scope for Customer LU

String ver= db(“fabric”).fetch("select
version(‘Customer’)").firstValue().

toString();
```

21.2.2 execute

Function: DBExecute	
Format	void execute(String sql, Object... params) void execute(Db.Statement statement, Object... params)
Description	Execute an SQL statement on the specified database.
Input	Sql/command - The SQL/command to be executed params - Values used for the prepare statement
Output	None. In case of failure- the method throws exception.

Example	//Get instance from Fabric
	
	`String getCommand = "get " + LU_NAME + ".'" +UID + "'";`
	`Db ci = db("fabric");`
	`ci.execute(getCommand);`
	
	// Open transaction and insert record into LUI table
	`Db ci = db("fabric");`
	`ci.beginTransaction();`
	`Object [] params = new Object[]{row.cell(0), row.cell(1), row.cell(2)};`
	`ci.execute("INSERT INTO INV_SUMMARY(NO_OF_INVOICES, TOT_INV_BAL, STATUS) VALUES (?, ?, ?)", params);`
	`ci.commit();`

Server:getConnection

Function: getConnection	
Format	Connection getConnection(String interfaceName)
Description	This method is used to establish a database connection.
Input	InterfaceName – The interface name defined within the studio In-memory DB => ludb
Output	The method returns a Connection object, which represents a connection to a specific database. This object is used to query the database.

Example	//First option – LUDB connection
	`Connection con = getConnection("ludb");`
	
	//Second option – Source DB connection. PG_SOURCE is a name of a database defined in K2View Fabric Studio as described in Section 6.2.
	`Connection conPG = getConnection("PG_SOURCE");`
	
	//Result
	Con parameter can later be used for DBExecutePrepared or DBQueryPrepared

Server:DBGetLastErrorStr

Function: DBGetLastErrorStr	
Format	String DBGetLastErrorStr()
Description	Returns the last error which occurred in the process.
Input	NA
Output	Last error description

Example	//Set an SQL string
	`String sql = "select UDAC_DESC FROM YP_REF_UDAC WHERE UDAC_CODE = 'SRL'";`
	
	//Execute the SQL
	`Object newUdacCode = DBSelectValue("ORA_SOURCE", sql, valuesArr);`
	
	//Get the last error in case the previous statement failed (relevant only if the process was not terminated due to the error)
	`String error = DBGetLastErrorStr();`
	
	//return the error;

21.2.3 rejectRecord

Function: rejectRecord	
Format	Void rejectRecord ()
Description	Reject the specific record
Input	Description - Error message to be printed into the log
Output	NA

Example	//Set an SQL string
	`sql = "SELECT PRODUCT_ISSUE_NUM FROM YP_PRODUCT";`
	
	//Execute the SQL
	`String prod_issue_num = DBSelectValue("ludb", sql, null).toString();`
	
	//Call reject the record
	`if (prod_issue_num == "2008" || prod_issue_num == "2005")`
	`rejectRecord ("Trn product record was rejected");`
	
	//Result
	In case prod_issue_num eqauls to 2008 or 2005, the record is rejected and the message is written into the log.

21.2.4 rejectInstance

Function: rejectRecord	
Format	Void rejectInstance ()
Description	Reject the instance
Input	Description - Error message to be printed into the log
Output	NA

Example	//Set an SQL string
	`sql = "SELECT PRODUCT_ISSUE_NUM FROM YP_PRODUCT";`
	
	//Execute the SQL
	`String prod_issue_num = DBSelectValue("ludb", sql, null).toString();`
	
	//Call reject the record
	`if (prod_issue_num == "2008" || prod_issue_num == "2005")`
	`rejectInstance ("Trn product record was rejected");`
	
	//Result
	In case prod_issue_num eqauls to 2008 or 2005, the instance is rejected and the message written into the log.

21.2.5 reportUserMessage

Format	Void reportUserMessage (String message)
Description	Displays a user message
Input	Description - Error message to be printed into the log
Output	NA

Example	//Set an SQL string
	`sql = “SELECT PRODUCT_ISSUE_NUM FROM YP_PRODUCT”;`
	
	//Execute the SQL
	`String prod_issue_num = DBSelectValue("ludb", sql, null).toString();`
	
	//Present a user message
	`if (prod_issue_num == “2008” || prod_issue_num == “2005”)`
	`reportUserMessage("Product issue number has incorrect value: ”||prod_issue_num);`
	
	//Result
	In case prod_issue_num equals to 2008 or 2005- a message is written into the log.

21.3 Translation

21.3.1 getTranslationValues

Function: getTranslationValues	
Format	Map <String><String>(String translationName, Object[] inputs)
Description	Returns the translation output parameters based on the inputs
Input	The translation input fields
Output	The translation output fields

Example	//trnProduct definition
	PRD_CD_IN, VER_IN, PRD_CD_OUT
	MX 1 MX1_NEW
	MX 2 MX2_NEW
	NB 1 NB1_NEW
	
	//Function code
	//Populate the translation input array
	`String[] trnValues = {“MX”, "2"};`
	
	//Call the translation function
	`Map trn = getTranslationValues("trnProduct", trnValues);`
	
	//Get PRD_CD_OUT from the returned map object
	`String new_product_code = trn.get("PRD_CD_OUT");`
	
	//Return the value
	`return new_product_code;`
	
	//Result
	`New_product_code = MX2_NEW`

21.4 Utilities

21.4.1 getInstanceID

Function: getInstanceID	
Format	String getInstanceID ()
Description	Returns the ID of the executed instance
Input	NA
Output	The instance ID

Example	//Populate customer_id with the instance ID
	`String customer_id = getInstanceID();`
	
	//Return the value
	`return customer_id;`

21.4.2 getNodeId

Function: getNodeId	
Format	String getNodeId ()
Description	Returns the Cassandra node ID. When executed in the studio (debug mode), it returns a constant string STUDIO-NODE-ID.
Input	NA
Output	The instance ID
Example	// Get the Node ID String nodeId = inDebugMode() ? "TEST_NODE" : getNodeId().toString();

22 Appendix B: Product Functions

22.1 Overview

These functions are part of the standard kit installation used to perform standard data manipulation tasks, and are visible on maps and in all objects that use a Java editor window.

These functions are part of the product, and are not editable.

> **Note:** In case new Product functions are added or existing product functions are updated, users can update the local copy. Right-click **Functions** in the project tree and select **Reset Product Functions**.

22.2 Date

22.2.1 k2_breakDate

Function: k2_breakDate	
Format	`Object[] k2_breakDate(String dateStr, String format)`
Description	Breaks a given date into a year, month, day, hour, minute, second and millisecond.
Input	Date and Date Format
Output	Array of year, month, day, hour, minute, second and millisecond.

Function: k2_breakDate	

Example	// Define a select to retrieve a date field from a table
	`String sql1 = "select CTCR_BATCH_PROG_DT FROM YP_CUSTOMER";`
	
	// Run the select on the table
	`Object date = DBSelectValue("ludb", sql1, null);`
	
	// Get array of the year, month, day, hour, minute, second and millisecond.
	`Object[] date_arr = (Object[])k2_breakDate (date.toString(), "yyyy-MM-dd");`
	
	//Example
	if date = “2014-12-02”.
	then date_arr array contains the following values-
	Year- 2014
	Month- 12
	Day-2
	Hour- 0
	Minute- 0
	Second- 0
	Millisecond- 0

22.2.2 k2_currentDate

Function: k2_currentDate	
Format	`String k2_currentDate()`
Description	Returns current date in the following format: "yyyy/MM/dd"
Input	NA
Output	Current date in following format: "yyyy/MM/dd"

Example	// Call function on 2-Dec-14-
	
	`String date = k2_currentDate();`
	
	//Result
	“2014/12/02”

22.2.3 k2_currentDateTime

Function: k2_currentDateTime	
Format	`String k2_currentDateTime()`
Description	Returns current date in the following format: " yyyy/MM/dd HH:mm:ss"
Input	NA
Output	Current date in in the following format: "yyyy/MM/dd HH:mm:ss"

Example	// Call function on 2-Dec-14 at 17:40
	`String date = k2_currentDate();`
	//Result
	`“2014/12/02 17:40:30”`

22.2.4 k2_currentTimeStamp

Function: k2_currentTimeStamp	
Format	`String k2_currentTimeStamp()`
Description	Returns current date in the following format: "yyyy/MM/dd HH:mm:ss:S"
Input	NA
Output	Current date in in the following format: ""yyyy/MM/dd HH:mm:ss:S"

Example	// Call function on 2-Dec-14 at 17:40
	`String date = k2_currentDate();`
	//Result
	`“2014/12/02 17:40:30:01”`

22.3 FileSystem

22.3.1 k2_Createfile

Function: k2_Createfile	
Format	`void k2_Createfile(String i_fileContent, String i_fileName, String i_location, Integer i_append)`
Description	Creates a physical file based on an input string.
Input	i_fileContent- the content of the file (String that needs to be written to the file) i_fileName- file name i_location- full path directory i_append- append indicator- can be 0 or 1 0- override a previous content of the file if exists 1- append the input content to previous content of the file if exists
Output	NA

22.3.2 k2_find_files

Function: k2_find_files	
Format	`Object[] k2_find_files(String path, String file_regExp)`
Description	Get list of all files matching file_regExp in a specific path
Input	String path- directory of the files String file_regExp- an expression which exists into the file name
Output	An array with a list of file names

Example	// Find all file names which contains 'Testing' under /home/k2view/testFiles
	
	`Object[]  fileList = k2_find_files ("/home/k2view/testFiles", "Tesing");`
	
	//Result
	All files containing "Testing" in the file name are returned by array.

22.4 Math

22.4.1 k2_abs

Function: k2_abs	
Format	`Object k2_abs(Double i_num)`
Description	Return an absolute value of the number
Input	A number
Output	Absolute value of the input number
Example	Send -3.45 as a parameter. The output is 3.45.

22.4.2 k2_ceil

Function: k2_ceil	
Format	`Double k2_ceil(Double value)`
Description	Round fractions up - returns the next highest integer value by rounding up value if necessary.
Input	A number
Output	A rounded number
Example	Input value = 4.75 Output value = 5.0

22.4.3 k2_floor

Function: k2_floor	
Format	`Double k2_floor(Double value)`
Description	Round fractions down - returns the next lowest integer value by rounding down value if necessary.
Input	A number
Output	A rounded number
Example	Input value = 4.75 Output value = 4.0

22.4.4 k2_minus

Function: k2_minus	
Format	`Double k2_minus(Double i_num1, Double i_num2)`
Description	Minus second parameters from first parameter
Input	2 numbers
Output	The result of i_num1- i_num2
Example	Input parameters- 3 and 2 Result = 1.0

22.4.5 k2_mod

Function: k2_mod	
Format	`Integer k2_mod(Integer i_number, Integer i_mod)`
Description	Modulo of a number
Input	2 numbers
Output	Returns the result for i_number % i_mod;
Example	Input parameters- 7 and 3 Result = 1

22.4.6 k2_multiply

Function: k2_multiply	
Format	`Double k2_multiply(Double i_num_1, Double i_num_2)`
Description	Multiply two numbers to get a result
Input	2 numbers
Output	Return i_num_1 * i_num_2
Example	Input parameters- 7 and 3 Result = 21.0

22.4.7 k2_plus

Function: k2_plus	
Format	`Double k2_plus(Double i_num1, Double i_num2, Double i_num3, Double i_num4, Double i_num5)`
Description	Sum up to five parameters to get a result
Input	5 numbers. One or more of the input parameters can be null. In this case- the function treats it as a zero.
Output	Accumulation of all input parameters
Example	Double res = k2_plus(1, 2, 3, null, null) // Result-6

22.4.8 k2_round

Function: k2_round	
Format	`Double k2_round(Double value, Integer precision)`
Description	Rounds a float - Returns the rounded value of the input number to a specified precision (number of digits after the decimal point). Precision can also be negative or zero.
Input	Double value – a number (double) Integer precision- defines the number of digits after the decimal point of the rounded result. This parameter must be populated by user.
Output	Rounded value of the input number- rounded up.

Example	Double res = k2_round(Double(2.345), 2);
	// Result
	//res = 2.35

22.5 String

22.5.1 k2_concat5

Function: k2_concat5	
Format	`String k2_concat5(String i_str1, String i_str2, String i_str3, String i_str4, String i_str5, String i_delimiter)`
Description	Concatenate up to 5 strings with a defined delimiter.
Input	5 Strings and a delimiter (6th parameter). One or more of input Strings can be null. Also, a delimiter can be null. In this case- all Strings are concatenated without a delimiter.
Output	Concatenated String

Example	`String concatName1 = k2_concat5("AA", "BB", "CC", "DD", "EE", "-");`
	`String concatName2 = k2_concat5("AA", "BB", "CC", "DD", "EE", null);`
	`String concatName3= k2_concat5("AA", "BB", null, null, null, ";");`
	
	// Results
	`// concatName1= "AA-BB-CC-DD-EE"`
	`// concatName2= "AABBCCDDEE"`
	`// concatName3 = "AA;BB"`

22.5.2 k2_ifNull

Function: k2_ifNull	
Format	`String k2_ifNull(String input, String value)`
Description	Sets a default value for an input String. When the input String is null- the function returns the input value entered, else returns the input itself.
Input	String input- input String String value- a default value which is returned if input String is null
Output	If String input is null, the function returns String value. Else- it returns String input.

Example	// SELECT CUSTOMER_TYPE for a specific customer. If selected CUSTOMER_TYPE field is empty (null), then replace it by a default value “IND” (individual customer)
	
	// Define a select to retrieve a date field from a table
	`String sql1 = "select CUSTOMER_TYPE FROM YP_CUSTOMER";`
	
	// Run the select on the table
	`Object res = DBSelectValue("ludb", sql1, null);`
	`String custType = String.valueOf(res);`
	
	`String defCustType=”IND”;`
	`String correctCustType = k2_ifNull(custType, defCustType);`
	
	// If custType is “BUS”, then function returns “BUS”, but if it is empty- function returns “IND”.

22.5.3 k2_ltrim

Function: k2_ltrim	
Format	`String k2_ltrim(String i_str, String i_charList)`
Description	Strip whitespace or other characters from the beginning of a string.
Input	String i_str- String to remove characters from. i_charList- the characters to remove, **null** treated as whitespace.
Output	The stripped String

Example	`String str1= k2_ltrim("yxabc  ", "xyz");`
	`String str2= k2_ltrim("     abc", null);`
	
	// Results-
	`// str1 = "abc  "`
	`// str2 = "abc"`

22.5.4 k2_pad

Function: k2_pad	
Format	`k2_pad(String i_str, String i_pad_string, Integer i_pad_length, Boolean i_padToRight)`
Description	Pad the i_pad_string to i_str according to i_pad_length.
Input	String i_str - String to pad out. String i_pad_string- String to pad with, **null** or *empty* treated as single space. Integer i_pad_length- size to pad to. Boolean i_padToRight- if a parameter is set to true or null- then the function performs a right pad. Else- if it is false – the function performs a left pad.
Output	Right or left padded String or original String if no padding is necessary.

Example	`String str1= k2_pad("bat", "x", 5, true);`
	`String str2= k2_pad("bat", "x", 3, true);`
	`String str3= k2_pad("bat", "x", 5, false);`
	
	//Results-
	`// str1= "batxx"`
	`//str2= "bat"`
	`//str3= "xxbat"`

22.5.5 k2_regexp_match

Function: k2_regexp_match	
Format	`Boolean k2_regexp_match(String pattern, String subject)`
Description	Perform a regular expression match
Input	String pattern- regular expression String subject- checked String
Output	true/false
Example	`Boolean b= k2_regexp_match ("Welcome(.*)", "Welcome to` K2View Fabric`");` // Results – b is true

22.5.6 k2_regexp_replace

Function: k2_regexp_replace	
Format	`String k2_regexp_replace(String pattern, String replacement, String subject)`
Description	Perform a regular expression search and replace
Input	String pattern- regular expression String replacement- replacement String. The Pattern is replaced by the String replacement. String subject- checked String
Output	Modified String
Example	String str = k2_regexp_replace("Welcome", "Hello", "Welcome to K2View Fabric"); // Result- "Hello to K2View Fabric"

22.5.7 k2_rtrim

Function: k2_rtrim	
Format	`String k2_rtrim(String i_str, String i_charList)`
Description	Strip whitespace or other characters from the end of a string
Input	String i_str- input String String i_charList- the set of characters to remove, null treated as whitespace
Output	The stripped String

Example	`String str1= k2_rtrim("  abc  ", null);`
	`String str2= = k2_rtrim("  abcyx", "xyz");`
	
	// Results-
	`//str1= "  abc"`
	`//str2= "  abc"`

22.5.8 k2_strpos

Function: k2_strpos	
Format	Integer k2_strpos(String str, String substring)
Description	Find position of the first occurrence of a substring
Input	String str- input String String substring- input substring
Output	Start position of the substring
Example	k2_strpos("TEST","ST") returns 3

22.5.9 k2_strrpos

Function: k2_strrpos	
Format	`Integer k2_strrpos(String str, String substring)`
Description	Find position of the last occurrence of a char in a string
Input	String str- input String String substring- input substring
Output	Start position of the substring
Example	k2_strrpos("TEST","T") returns 4

22.5.10 k2_strtolower

Function: k2_strtolower	
Format	`String k2_strtolower(String i_str1)`
Description	Converts a String to lowercase
Input	Input String
Output	Lowercase String
Example	k2_strtolower(“TEST”) returns “test”

22.5.11 k2_strtoupper

Function: k2_strtoupper	
Format	`String k2_strtoupper(String i_str1)`
Description	Converts a String to uppercase
Input	Input String
Output	Uppercase String
Example	k2_strtoupper(“test”) returns “TEST”

22.5.12 k2_trim

Function: k2_trim	
Format	`String k2_trim(String i_str, String i_charList)`
Description	Strip whitespace or other characters from the beginning and end of a string
Input	String i_str- input String String i_charList- the set of characters to remove, null treated as whitespace
Output	The stripped String

Example	`String str1= k2_trim(“  abc  “, null);`
	`String str2= = k2_rtrim(“  abcyx”, “xyz”);`
	
	// Resutls-
	//str1= “abc”
	//str2= “ abc”

22.6 Utilities

22.6.1 k2_getInstanceId

Function: k2_getInstanceId	
Format	`String k2_getInstanceId()`
Description	Get the current instance id
Input	NA
Output	Instance id (Primary Key) of the Logical Unit
Example	When running a Logical Unit on customer 1234, the function returns 1234

22.6.2 k2_IF

Function: k2_IF	
Format	String k2_IF(String i_var, String i_var_check_val, String i_var_true_val, String i_var_false_val)
Description	Compare between values of 2 input parameters. If they are equal- return the input true value. Else- return the input false value.
Input	String i_var- first input parameter String i_var_check_val- compared value String i_var_true_val- this value is returned if i_var equals to i_var_check_val String i_var_false_val- this value is returned if i_var does not equal to i_var_check_val
Output	i_var_true_val or i_var_false_val

Example	String sql1 = "select CUSTOMER_TYPE FROM YP_CUSTOMER";
	
	// Run the select on the table
	Object res = DBSelectValue("ludb", sql1, null);
	String custType = String.valueOf(res);
	
	// custType = “BUS”
	String flag = k2_IF(custType, “IND”, “Individual customer”, “Business customer”);
	
	// Result- “Business customer”

22.6.3 New JDBC Wrapper

A simplified syntax of commands to invoke data bases to make command more intuitive and work more efficiently.

> **Note:** you can use the new commands in your code, the generate code option of Fabric Studio still generates the old commands for DB access.

22.6.4 Run queries against a Fabric LU

ludb(<LU Type> , <Instance ID>).fetch("select A from a_table where col1=?", "VALUE").forEach(row->{

assert(row.get(0)==row.get("A")); // getting column by name runs efficiently (data not copied into a map), and encouraged in most cases

assert(row.get(0)==row.cells()[0]);

assert(row.cells()[0]==row.cell(0));

});

Example:

ludb("LittlePony", whichPony).fetch("SELECT * from PONY inner join PONY_HAS ON PONY.NAME=PONY_HAS.NAME").forEach(record->{…

From Fabric 4.0 and on, in order to work with fabric_local a new command was introduced called ludbLocal.

22.6.5 Run queries against the LU you are in the context of (migration, sync)

ludb().fetch("select * ...").forEach(r->{ ...});

22.6.6 Run queries against a logical database interface:

db("<interface name>").fetch(...).forEach(r->{ ...

22.6.7 New JDBC Wrapper for Web Services- Example

```
String sql = "SELECT SUBSCRIBER_NO, BILL_SEQ_NO FROM BILL";
// LU name must be
Db.Rows rows = ludb(<LU Name>, <instanceID>).fetch(sql, <val1>, <val2>, ...);
// Loop on the results
rows.forEach(row->{
            row.get("SUBSCRIBER_NO");
           row.get(1); // get BILL_SEQ_NO
});
```

22.6.8 Handle Loops

Use a good old fashion loop:

```
for (Db.Row row:db(..).fetch(...)) {...}
```

Break a loop down further:

```
Db db =  db(..);
for (Db.Row row:db.fetch(...)) {… }
```

To get the first row only, use:

```
db(...).fetch(...).firstRow()
```

If no rows are returned, the first row returns an empty Row (not null) so it is safe to do:

```
Object value = db(...).fetch(...).firstRow().get("COL_7");
```

To get only one value - the first value of the first row, use:

```
Object value = db(...).fetch(...).firstValue();
```

Returns null if no value is returned.

To iterate all columns:

```
...fetch(...).forEach(r->{
   // each r is a map that represents a row.
   // Iteration order is guaranteed to follow the column order.
      r.forEach((col,value)->{
            ...
      });
});
```

DB Execute command:

```
db(...).execute("SQLEXEC % %", a, b);
```

22.6.9 DbTemplate

A DbTemplate is a class that can generate a dynamic XML from a template.

It is very useful when you want to customize the response of a Fabric web service, based on the output from a set of SQL queries.

It was designed to support the hierarchical nature of XML, so a query can use any "outer level" query result as an argument.

22.6.9.1 DbTemplate- Features

- Execute a query on the LUDB and iterate over its results.
- Recursively generate nested XML structures.
- Customizable behavior on null values.
- Accept external input as variables.
- Use variables in queries.
- Queries can be executed on any interface (not just Fabric).
- Support prepared statements.
- Support string formatting codes (%s, %.2f,%d...).
- Support DISTINCT keyword to eliminate duplicated rows from the results of a SELECT statement.
- Support GROUPBY clause in collaboration with the SELECT statement to organize identical data into groups.

22.6.9.2 DbTemplate - XML Content- Example

```
<?xml version="1.0"?>
<Outer>
    [[[pat =  select PATIENT_ID,FIRST_NAME, LAST_NAME from
patient
       <Inner>
       <ID>${pat.PATIENT_ID}</ID>
             <FirstName>${pat.FIRST_NAME}</FirstName>
             <FirstLast>${pat.LAST_NAME}</FirstLast>
       </Inner>
    ]]]
</Outer>
```

22.6.9.3 DbTemplate- execute() Function

execute() function has 4 parameters:

- Custom string for lookup failure
- Custom string for null value
- Map with a list of input parameter for the dbTemplate
- Template type. Valid values for templateType: “Xml”, “Json”

22.6.9.4 DbTemplate - Usage Examples

> **Note:** You need to put the XML file as part of the LU on the Studio under Resource Files folder

22.6.9.4.1 Basic usage

```
DBExecute("fabric","get Customer." + iid,null);
DbTemplate template = new
DbTemplate("customer.xml",queryExecutor);
return template.execute(templateType);
```

Valid values for templateType: “Xml”, “Json”

22.6.9.4.2 Send Input Variables

It is possible to pass over external variables (as a java map) and use them in the xml.

```
DBExecute("fabric","use Album.*",null);
Map<String,Object> map = new HashMap<>();
map.put("i_Date", new Date());
map.put("i_TitlePrefix", "D");
map.put("i_maxTrackTimeMinutes",2);

DbTemplate template = new
DbTemplate("album.xml",queryExecutor);
return template.execute(null, "", map, templateType);
```

22.6.9.4.3 Custom String for Null Values

By default, when a variable resolves to null, the DbTemplate skips the entire line.

However, you could use the @ character inside a variable name in order to allow nulls and replace them with empty strings.

For example:

```
<VISIT>
    <visit_id>${@visits.visit_id}</visit_id>
    <visit_date>${@visits.visit_date}</visit_date>
    <status>${@visits.status}</status>
</VISIT>
```

If you want to use value other than null, you can pass it to the ‘execute’ function call.

In the example below, we would like to use the string EMPTY as a placeholder for null values.

```
DBExecute("fabric","get Customer." + iid,null);
DbTemplate template = new
DbTemplate("customer.xml",queryExecutor);
return template.execute(null, "EMPTY", null, templateType);
```

22.6.9.4.4 Custom String for Lookup Failure

By default, when a lookup fails, the dbTeplate skips the entire line.

However, you could use the @ character inside a variable name in order to allow a lookup failure and replace it with a customized string.

If you would like to use other value than null, you can pass it to the 'execute' function call.

In the example below, we would like to use the string EMPTY as a placeholder for a lookup failure.

```
DBExecute("fabric","get Customer." + iid,null);
DbTemplate template = new
DbTemplate("customer.xml",queryExecutor);
return template.execute("EMPTY", null, null, templateType);
```

22.6.9.4.4.1 String Formatting Code

You can add formatting to the output string by adding ':' to the variable name.

For example- you can define a Real attribute in both ways:

- <REAL>${Patient.REAL}</REAL>
- <REAL>${Patient.REAL:%.0f}</REAL>

For the first option- the output will be <REAL>8.0</REAL>. However, for the second option the output will be <REAL>8</REAL>.

22.6.9.5 Comparison Operators

You can add define a block that will be presented only in case the conditions in the comparison operator are meet.

The DbTemplate supports 3 types of operators:

- ALL- this operator returns true if all of the defined conditions are met.
- ANY- this operator returns true if at least one of the conditions is met.
- NONE- this operator returns true if none of the defined conditions is met.

The following syntax needs to be set to check operators into the xml template file:

```
<Operator> {"col1_name":col1_val,"col2_name":col2_val}
```

You can check columns of outer and inner queries. In order to include columns of different queries into one operator, you need to add the query name to the checked column. For example:

- ALL{"patient.PATIENT_NAME": "David", :visit.VISIT_ID":2}

You can also check for null values in your condition. For example:

- ANY {"PATIENT_ID":4,"FIRST_NAME":null}

> **Note:** You cannot nest several operators in your DbTemplate. For example- the following condition is not supported:
>
> ANY{NONE{"PATINET_NAME":"David"}, ALL{"VISIT_ID":6}}

22.6.9.6 The EMPTY Directive

You can define a block of code that will be presented only if the outer query returns zero results. This block of code can contain all the DbTemplate directive types, including more EMPTY directives.

22.6.9.6.1 EMPTY Directive- Example

```
[[[alr = select ALERGY_ID from P1.ALLERGIES
 <Inner>
<ID>${alr.ALERGY_ID}</ID>
 </Inner>
]]]
[[[EMPTY
  [[[pat = select PATIENT_ID from P7.patient
    <Inner>
<ID>${pat.PATIENT_ID}</ID>
    </Inner>
  ]]]
]]]
```

22.6.9.7 Set Default Value For Null and Empty String Result

You can add a specific default value to a variable by adding ';' and the requested value after the variable name in the template. For example:

- `<FirstName>${res.FIRST_NAME;default_value}</FirstName>`

> **Note**: This value will not be taken in case of lookup failure.

22.6.9.8 Execute Fabric Commands

You can also execute fabric commands inside the xml template file.

For example:

```
[[[pat = select PATIENT_ID, FIRST_NAME from P7.patient
  [get P1.${pat.PATIENT_ID}]
]]]
```

22.6.9.9 DISTINCT Keyword

The DISTINCT keyword eliminates duplicate rows from the results of a SELECT statement.

22.6.9.9.1 DISTINCT Keyword- Example

```
[[[ Patient = select SSN, PATIENT_ID, FIRST_NAME, LAST_NAME, STATE_B, STATE
from PATIENT
        <Patient>
            <Patient_id>${Patient.PATIENT_ID}</Patient_id>
            <VISITS>
            [[[ visits = select visit.patient_id, visit_id, visit_date, status from PATIENT
left join visit on PATIENT.PATIENT_ID = visit.patient_id
                [[[ DISTINCT {visits.status }
                    <DISTINCT>

    <DISTINCT_visit_id>${@visits.visit_id}</DISTINCT_visit_id>

    <DISTINCT_STATUS>${@visits.status}</DISTINCT_STATUS>
                 </DISTINCT>
                ]]]
            ]]]
            </VISITS>
        </Patient>
    ]]]
```

The example above returns the following output:

```
<Patient>
    <Patient_id>294</Patient_id>
      <VISITS>
            <DISTINCT>
                <DISTINCT_visit_id>1540</DISTINCT_visit_id>
                <DISTINCT_STATUS>CLOSED</DISTINCT_STATUS>
            </DISTINCT>
            <DISTINCT>
                <DISTINCT_visit_id>1542</DISTINCT_visit_id>
                <DISTINCT_STATUS>SCHEDULED</DISTINCT_STATUS>
             </DISTINCT>
            <DISTINCT>
                <DISTINCT_visit_id>1543</DISTINCT_visit_id>
                <DISTINCT_STATUS>CANCELLED</DISTINCT_STATUS>
```

```
                    </DISTINCT>
                </VISITS>
            </Patient>
```

22.6.9.10 GROUPBY Clause

The GROUPBY clause is used in collaboration with the SELECT statement to arrange identical data into groups.

> **Note:** the GROUPBY need to be done on column with order by statement

22.6.9.10.1 GROUPBY Clause- Example

```
 [[[ Patient= select SSN, PATIENT_ID, FIRST_NAME, LAST_NAME, STATE_B, STATE
from PATIENT
<Patient>
    <Patient_id>${Patient.PATIENT_ID}</Patient_id>
    <VISITS>
    [[[ visits = select visit.patient_id, visit_id, visit_date, status from PATIENT order by
PATIENT.PATIENT_ID
        [[[ GROUPBY {visit.patient_id}
            <VISIT>
                <visit_id>${@visits.visit_id}</visit_id>
                <status>${@visits.status}</status>
            </VISIT>
        ]]]
    ]]]
    </VISITS>
]]]
```

The example above returns the following output:

```
<Patient>
    <Patient_id>898</Patient_id>
    <VISITS>
            <VISIT>
                <visit_id>1540</visit_id>
                <status>CLOSED</status>
            </VISIT>
            <VISIT>
                <visit_id>1541</visit_id>
                <status>CLOSED</status>
            </VISIT>
            <VISIT>
                <visit_id>1542</visit_id>
                <status>OPEN</status>
            </VISIT>
    </VISITS>
</Patient>
```

22.6.9.11 Template Example

```
<?xml version="1.0"?>
<albums>
    [[[album  =  select AlbumId,Title, artistId as artist from albums
where title like "${i_TitlePrefix}%"
    <album id=${album.AlbumId}>
        <Title>
            <TitleText>${@album.Title}</TitleText>
            <Date>${i_date}</Date>
        </Title>
        [[[  artist (album.Artist ) = select Name from artists where
ArtistId = ?
        <artist id=${album.artist}>
            <name>${artist.Name}</name>
        </artist>
        ]]]after the closing tag you can write comments - it is ignored
by the parser.

        [[[track(album.AlbumId,i_maxTrackTimeMinutes) = select
TrackId,Name,Milliseconds/1000 as Seconds from tracks where AlbumId = ?
and Milliseconds < 60000 * ?
        <track>
            <ID>${track.TrackId}</ID>
            <Name>${track.Name}</Name>
            <Seconds>${track.Seconds}</Seconds>
        </track>
        ]]]
        [[[  empty = select * from artists where ArtistId =
1234323453455345
        <HiddenText>
        This text is not expected to be on the output, because
        the query "empty" above returns 0 rows.
        </HiddenText>
        ]]]
    </album>
    ]]]
</albums>
```

22.6.9.11.1 Output of DBTemplate Example

```
<?xml version="1.0"?>
<albums>
    <album id=25>
        <Title>
            <TitleText>Da Lama Ao Caos</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=18>
            <name>Chico Science & Nação Zumbi</name>
        </artist>

        <track>
```

```
            <ID>276</ID>
            <Name>Maracatu De Tiro Certeiro</Name>
            <Seconds>88</Seconds>
        </track>
        <track>
            <ID>279</ID>
            <Name>Risoflora</Name>
            <Seconds>105</Seconds>
        </track>
    </album>
    <album id=59>
        <Title>
            <TitleText>Deep Purple In Rock</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=58>
            <name>Deep Purple</name>
        </artist>

    </album>
    <album id=69>
        <Title>
            <TitleText>Djavan Ao Vivo - Vol. 02</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=80>
            <name>Djavan</name>
        </artist>

    </album>
    <album id=70>
        <Title>
            <TitleText>Djavan Ao Vivo - Vol. 1</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=80>
            <name>Djavan</name>
        </artist>

    </album>
    <album id=78>
        <Title>
            <TitleText>Deixa Entrar</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=83>
            <name>Falamansa</name>
        </artist>

        <track>
            <ID>975</ID>
```

```
            <Name>Deixa Entrar</Name>
            <Seconds>33</Seconds>
        </track>
        <track>
            <ID>983</ID>
            <Name>Principiando/Decolagem</Name>
            <Seconds>116</Seconds>
        </track>
    </album>
    <album id=98>
        <Title>
            <TitleText>Dance Of Death</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=90>
            <name>Iron Maiden</name>
        </artist>

    </album>
    <album id=161>
        <Title>
            <TitleText>Demorou...</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=108>
            <name>Mônica Marianno</name>
        </artist>

        <track>
            <ID>1968</ID>
            <Name>Demorou!</Name>
            <Seconds>39</Seconds>
        </track>
    </album>
    <album id=172>
        <Title>
            <TitleText>Diary of a Madman (Remastered)</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=114>
            <name>Ozzy Osbourne</name>
        </artist>

    </album>
    <album id=183>
        <Title>
            <TitleText>Dark Side Of The Moon</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=120>
            <name>Pink Floyd</name>
```

```
        </artist>

    </album>
    <album id=242>
        <Title>
            <TitleText>Diver Down</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=152>
            <name>Van Halen</name>
        </artist>

        <track>
            <ID>3054</ID>
            <Name>Cathedral</Name>
            <Seconds>82</Seconds>
        </track>
        <track>
            <ID>3056</ID>
            <Name>Intruder</Name>
            <Seconds>100</Seconds>
        </track>
        <track>
            <ID>3059</ID>
            <Name>Little Guitars (Intro)</Name>
            <Seconds>42</Seconds>
        </track>
        <track>
            <ID>3063</ID>
            <Name>Happy Trails</Name>
            <Seconds>65</Seconds>
        </track>
    </album>
    <album id=266>
        <Title>
            <TitleText>Duos II</TitleText>
            <Date>Thu Oct 27 22:37:38 IDT 2016</Date>
        </Title>
        <artist id=201>
            <name>Luciana Souza/Romero Lubambo</name>
        </artist>

    </album>
</albums>
```