API Gateway

Version 7.6.2

14 July 2020

Key Property Store User Guide

This documentation describes the following Axway software:

Axway API Gateway 7.6.2

Contents

Preface

This guide describes how to configure and manage the API Gateway Key Property Store (KPS). The KPS enables you to manage API Gateway data referenced from policies running on the API Gateway.

Who should read this guide

The intended audience for this guide is KPS administrators and policy developers. For more details on API Gateway user roles, see the *API Gateway Concepts Guide*. This guide assumes that you are familiar with the following:

- Database concepts such as tables, rows, and keys
- API Gateway configuration and deployment
- API Gateway selectors
- Using command line tools
- Database configuration where database storage is required

For more details on API Gateway configuration and selectors, see the *API Gateway Policy Developer Guide*.

How to use this guide

This guide should be used with the other guides in the API Gateway documentation set. Before you begin, review this guide thoroughly. The following is a brief description of the contents of each chapter:

Introduction to KPS on page 10 provides an overview of the KPS architecture and features.

Get started with KPS on page 13 explains how to develop an example KPS table for managing simple user information.

Configure KPS in Policy Studio on page 23 provides more detail on how to define general KPS configuration using the Policy Studio graphical tool.

Access KPS data using selectors on page 29 explains how to access data in policies on the API Gateway at runtime.

Manage KPS using the kpsadmin tool on page 31 explains how to manage a KPS, independent of data source.

Configure Apache Cassandra KPS storage on page 43 explains how to store KPS data in an external Apache Cassandra database.

Configure database KPS storage on page 44 explains how to store KPS data in a relational database (for example, Oracle, MySQL, IBM DB2, or Microsoft SQL Server).

Configure file-based KPS storage on page 56 explains how to store KPS data in a directory on the file system.

Related documentation

The AMPLIFY API Management solution enables you to create, publish, promote, and manage Application Programming Interfaces (APIs) in a secure and scalable environment. For more information, see the *AMPLIFY API Management Getting Started Guide*.

The following reference documents are also available on the Axway Documentation portal at https://docs.axway.com:

- *Supported Platforms*

  Lists the different operating systems, databases, browsers, and thick client platforms supported by each Axway product.

- *Interoperability Matrix*

  Provides product version and interoperability information for Axway products.

Support services

The Axway Global Support team provides worldwide 24 x 7 support for customers with active support agreements.

Email support@axway.com or visit Axway Support at https://support.axway.com.

See "Get help with API Gateway" in the *API Gateway Administrator Guide* for the information that you should be prepared to provide when you contact Axway Support.

Training services

Axway offers training across the globe, including on-site instructor-led classes and self-paced online learning. For details, go to: http://www.axway.com/support-services/training

Accessibility

Axway strives to create accessible products and documentation for users.

This documentation provides the following accessibility features:

Screen reader support

- Alternative text is provided for images whenever necessary.
- The PDF documents are tagged to provide a logical reading order.

Support for high contrast and accessible use of colors

- The documentation can be used in high-contrast mode.
- There is sufficient contrast between the text and the background color.
- The graphics have the right level of contrast and take into account the way color-blind people perceive colors.

Updates and revisions

This guide includes the following documentation changes.

Changes in version 7.6.2

- Added information on using the KPS scripting API. For details, see *Use the KPS scripting API* on page 58.
- Added information on the `kpsadmin diagnostics` command, which you can use to help diagnose common KPS and Apache Cassandra configuration issues. For more details, see *Manage KPS using the kpsadmin tool* on page 31.

Changes in version 7.6.1

No changes.

Changes in version 7.6.0

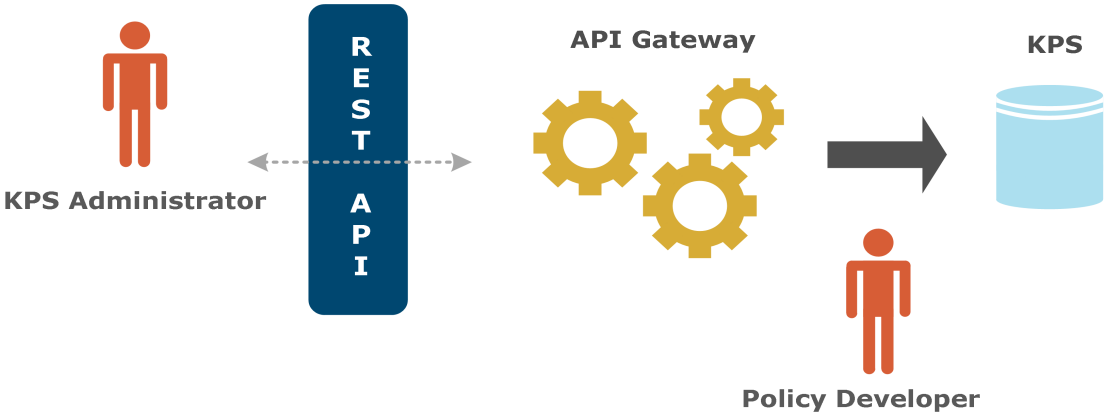
- Removed references to API Gateway server-side support for Windows.
- The getting started topic has been updated to explain that using `key` in a KPS property name causes deployment errors. For details, see *Get started with KPS* on page 13.

Introduction to KPS 1

A Key Property Store (KPS) is a table of data that can be referenced from policies running on the API Gateway. Data in a KPS table is assumed to be read frequently and seldom written, and can be changed without incurring an API Gateway service outage. KPS tables are shared across an API Gateway group.

KPS architecture

The following diagram shows a simple role-based architecture:



A KPS is typically used to store property values that are used in policies running on an API Gateway. KPS data is injected into policies using *selectors* that are first created in Policy Studio by *policy developers*. Selectors are evaluated and expanded dynamically at runtime. For example, a KPS table could contain authorization tokens for different users. A policy could look up the token for the current user and insert it into an HTTP request.

KPS tables are organized into *collections*. The tables in a collection typically have some sort of relationship to one another. For example, the OAuth collection contains a set of tables that store all OAuth-related data. Every KPS table is assigned an alias so that it can be easily referred to in a policy or a REST request. KPS collections and tables can be created by policy developers using Policy Studio.

KPS administrators can use the API Gateway Manager web console to view and modify KPS data at runtime. This is a business or operational role that manages dynamic policy configuration data in a KPS (for example, customer details, authorization levels, or quotas). This means that this information does not need to be configured at design time by policy developers.

For more details on API Gateway architecture, components, and roles, see the *API Gateway Concepts Guide*.

KPS data stores

KPS data can be stored in one of the following locations:

- **External Apache Cassandra database**:
  Data can be distributed across multiple nodes to provide high availability. This is the default data store. See *Configure Apache Cassandra KPS storage* on page 43.
- **Relational database**:
  Accessible to all API Gateway instances in the API Gateway group. See *Configure database KPS storage* on page 44
- **JSON files**:
  On the local file system (deprecated). See *Configure file-based KPS storage* on page 56

Note Custom KPS data defined in Policy Studio supports Cassandra, database, and file data stores. API Manager KPS data (Client Registry and API Catalog) supports Cassandra only. File-based KPS is deprecated and will be removed in a future release.

KPS client applications

API Gateway provides the following client applications:

- **Policy Studio**:
  Enables policy developers to create KPS collections and tables, and to configure data sources. See *Configure KPS in Policy Studio* on page 23.
- **API Gateway Manager**:
  Includes a visual web-based interface to enable KPS administrators to view and modify KPS data at runtime. See *Get started with KPS* on page 13
- **kpsadmin command**:
  Supports KPS data entry and other administrative functions. It is designed for use in a development environment. See *Manage KPS using the kpsadmin tool* on page 31
- **KPS REST API**:
  Enables remote programmatic clients to read and write KPS data. See the API Gateway REST API available from Axway Support at https://support.axway.com

When to use a KPS

KPS provides a flexible data storage service that can be used to store any configuration data. Its primary function is to make this data available to selectors at runtime, and it is optimized for this purpose. This makes it most suitable for data with the following characteristics:

- Data is common to all API Gateways in an API Gateway group. KPS is not suitable for data that is specific to one particular API Gateway.

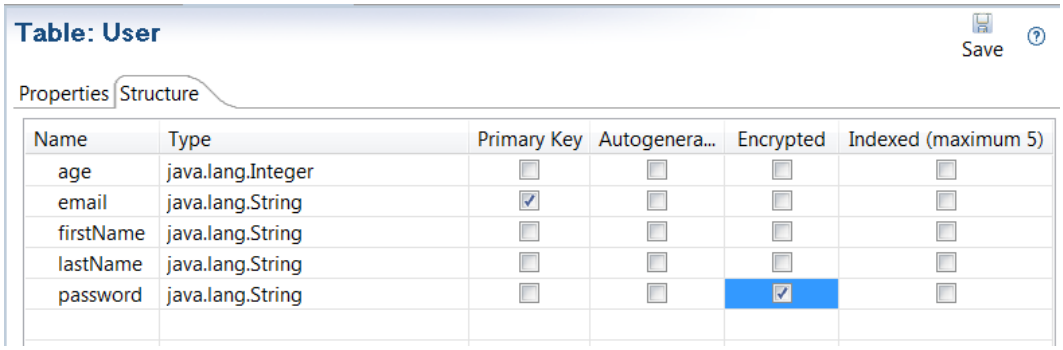
- The data schema is relatively simple. Each KPS table is assumed to be independent of all others, and referential integrity across tables is not supported.
- Data can change while API Gateways are running. Updating Cassandra-backed or database-backed KPS tables does not require an API Gateway restart. Changeable data should therefore be stored in KPS instead of hard-coded into policies.
- Queries always involve looking up a key value in a table to retrieve a single object. This is the usage model supported by selectors. Ad hoc queries that involve searching for non-key properties are not supported.
- Multi-operation transactions are not required. Each read or write to a KPS table is considered a standalone operation. Locking or rollback across multiple operations is not supported.

Get started with KPS 2

This topic explains how to develop an example KPS table for managing simple user information.

Example KPS table

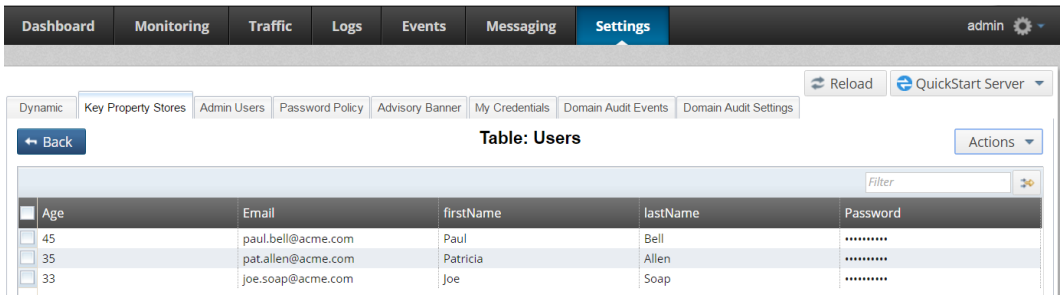
The final structure of the example table is displayed in Policy Studio as follows:



This table structure is described as follows:

Column	Type	Description
`age`	`java.lang.Integer`	User age.
`email`	`java.lang.String`	User email address. This is selected in Policy Studio as a unique **Primary Key**, which is indexed implicitly by default.
`firstName`	`java.lang.String`	User first name.
`lastName`	`java.lang.String`	User surname.
`password`	`java.lang.String`	User password, which is selected as **Encrypted** in Policy Studio.

Example table data is displayed on the **Settings** > **Key Property Stores** tab in the API Gateway Manager web console:



Before you begin

The following prerequisite steps apply to this example:

1. Ensure that an API Gateway and an Admin Node Manager are running.
2. Start Policy Studio, and connect to the Admin Node Manager.

For more details, see the *API Gateway Installation Guide*.

Define KPS configuration with Policy Studio

The main steps for configuring KPS tables in Policy Studio are as follows:

Step 1: Define where data will be stored on page 14

Step 2: Define the KPS table on page 15

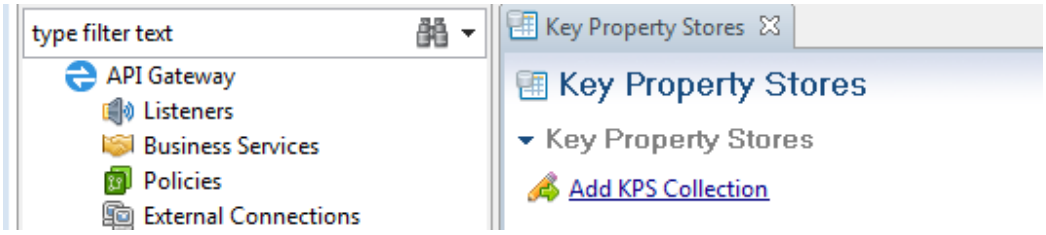
Step 3: Define a policy that accesses the table on page 17

Step 4: Deploy the configuration on page 19

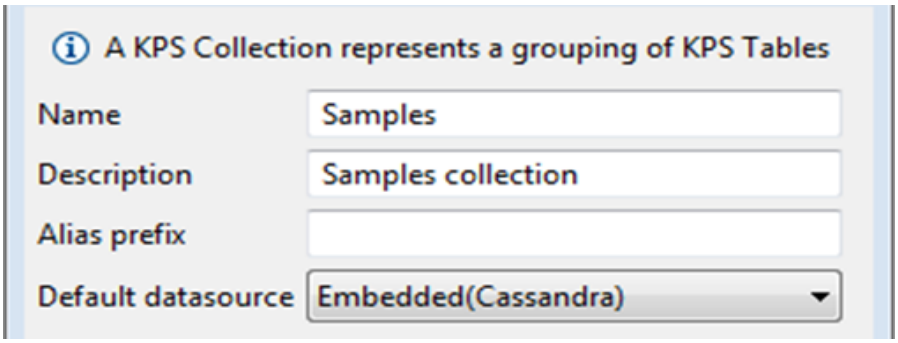
Step 1: Define where data will be stored

You must first create a KPS collection in which to store the KPS table configuration. Perform the following steps:

1. In the Policy Studio tree, select **Environment Configuration** > **Key Property Stores**, then select **Add KPS Collection**.



2. In the **Add KPS Collection** dialog, name the collection **Samples**.



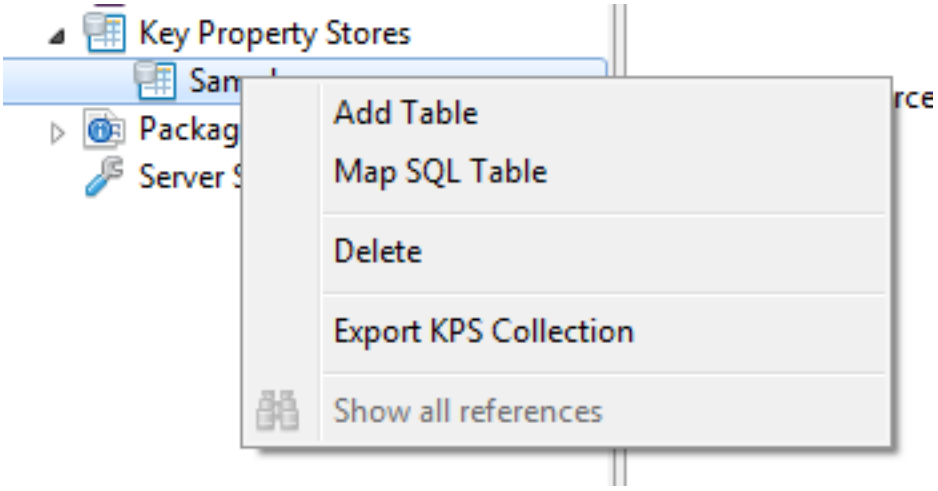
3. Select a **Default data source** of **Cassandra** for all tables in the collection.

   Note Leave the **Alias prefix** field blank.

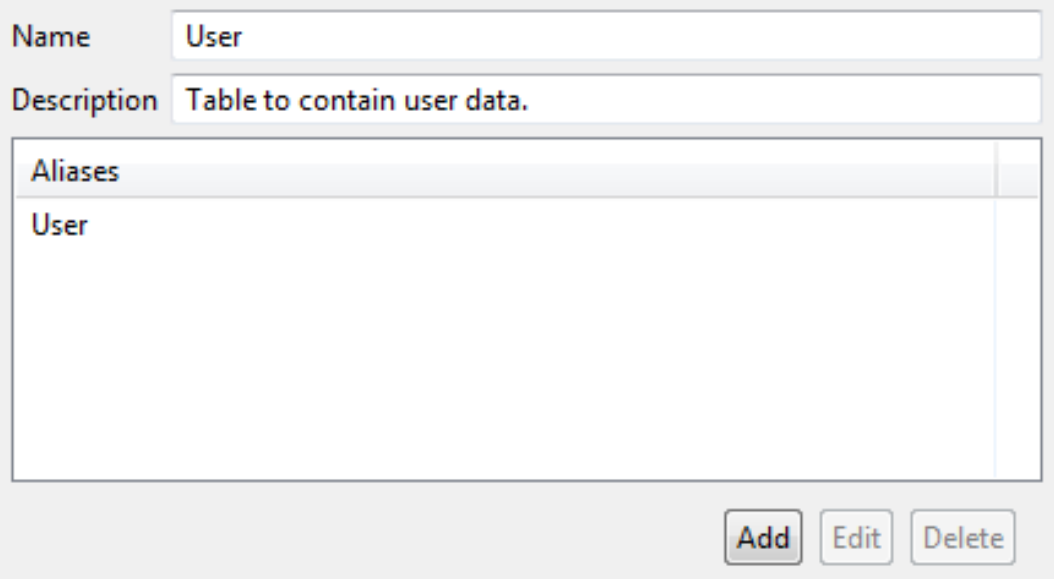
Step 2: Define the KPS table

To create a table, perform the following steps:

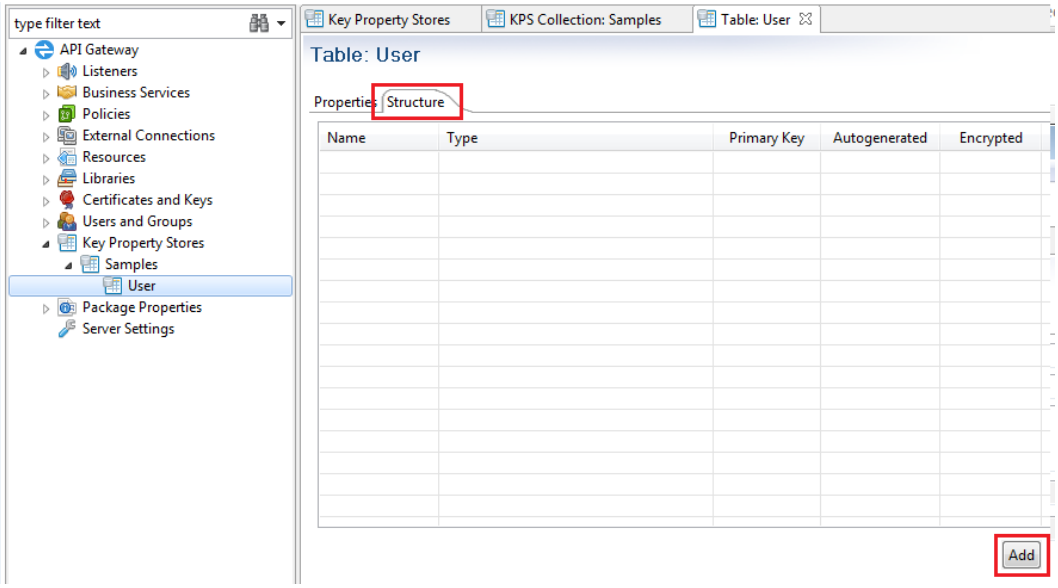
1. In the Policy Studio tree, right-click the newly-created **Samples** collection, and select **Add Table**:



2. In the dialog, enter a **Name** of `User`, and provide a **Description**.
3. Click **Add** to assign an alias of **User** to this table. A table must have at least one alias.



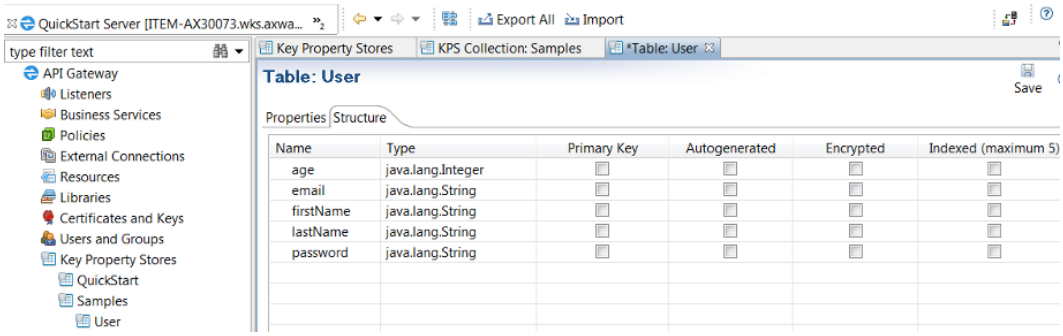
4. Next define the table structure. This consists of the table columns and the data type stored in each column. Select the **User** table and **Structure** tab, and click **Add**:



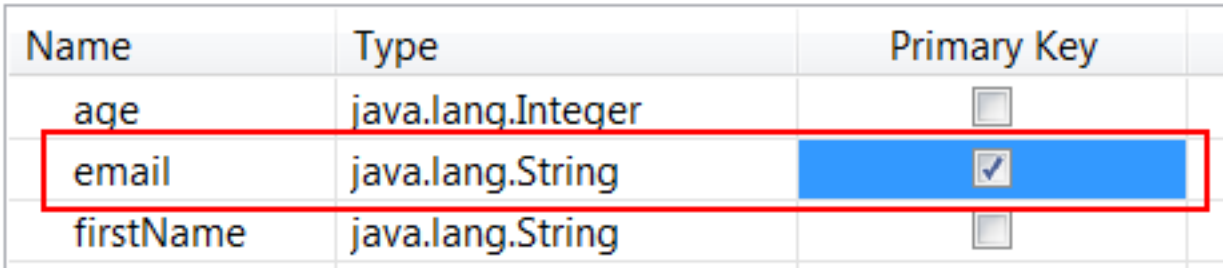
5. Repeat to add the following columns for your table structure in the **Add Property** dialog:
   - `email`
   - `password`
   - `firstName`
   - `lastName`
   - `age`

   Note `age` has an `Integer` (numeric) **Type**. All the other columns are `String`. For more details, see *KPS table structure* on page 26.

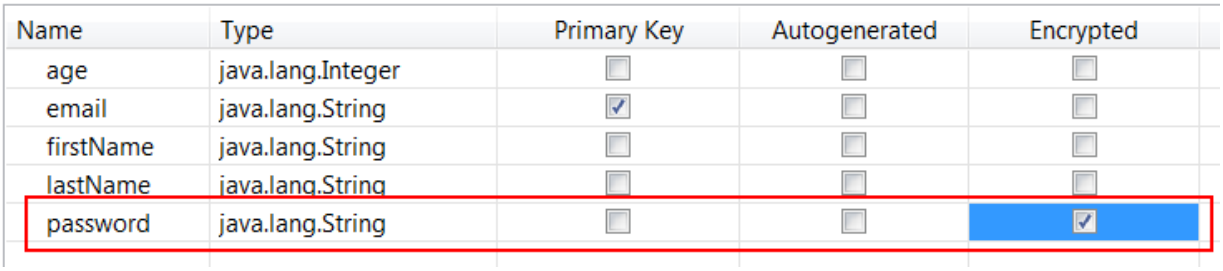
5. When you select the **User** table, you should have the following structure:



6. You want the **email** field to be the primary key for the table, so select **Primary Key** for this field.



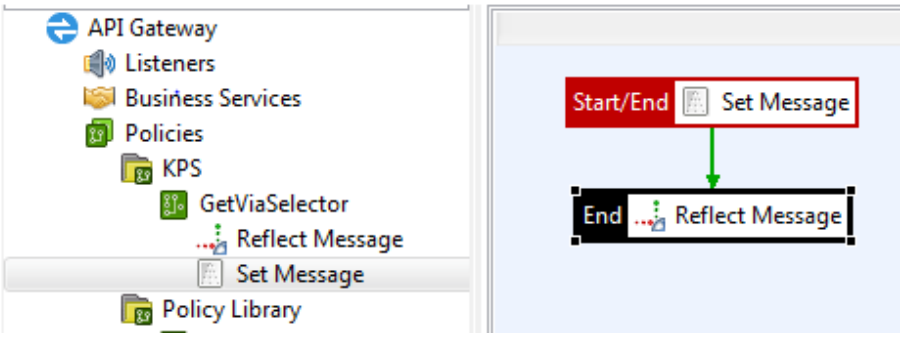
7. You want the **password** field to be encrypted when stored in the data source, so select **Encrypted** for this field.



Step 3: Define a policy that accesses the table

To define a test policy that accesses the table, perform the following steps:

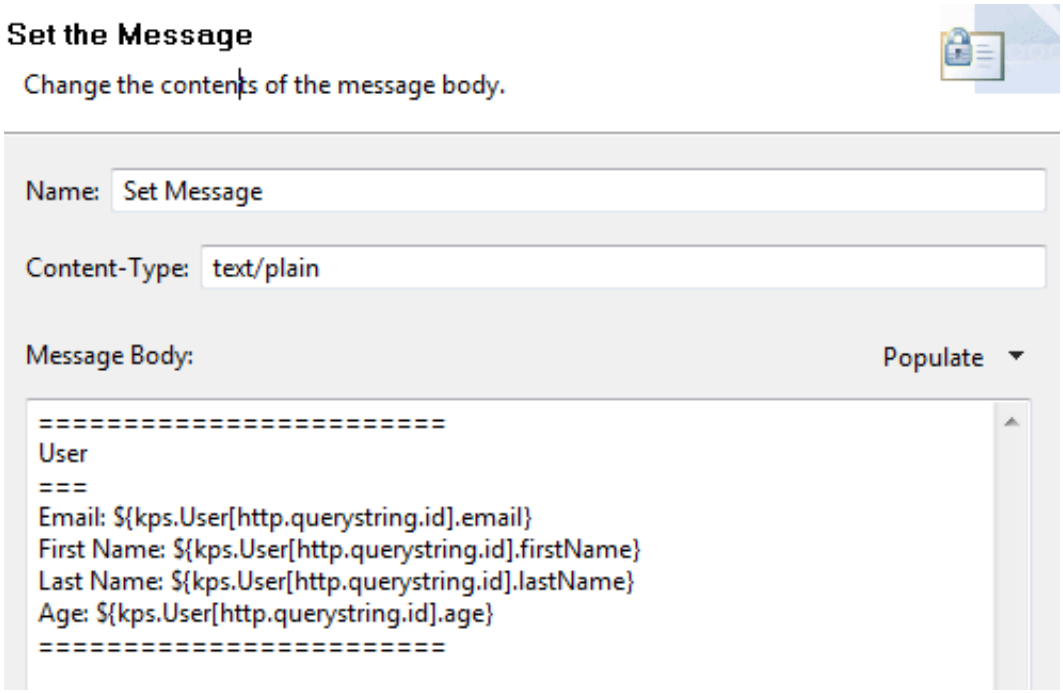
1. Add a test policy with a **Set Message** filter from the **Conversion** filter category.



2. Right-click the filter and set it as the **Start** filter for the policy.
3. Enter a filter **Content-Type** of `text/plain`.
4. Enter the following **Message Body** for use in the policy:

```
========================
User
===
Email: ${kps.User[http.querystring.id].email}
First Name: ${kps.User[http.querystring.id].firstName}
Last Name: ${kps.User[http.querystring.id].lastName}
Age: ${kps.User[http.querystring.id].age}
========================
```

These settings are displayed as follows in the **Set Message** filter:



The message body value are specified using selectors, which are evaluated and expanded dynamically at runtime. For example, the user age is specified using the following selector string:

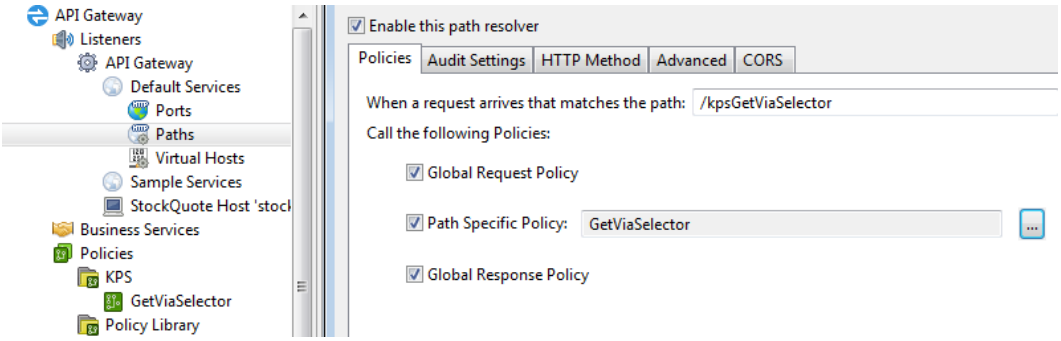
```
${kps.User[http.querystring.id].age}
```

The selector parts are explained as follows:

Selector part	Description
`${`	Indicates the start of the selector using a `{` bracket.
`kps`	Specifies that selector should query a KPS table.
`.User`	Specifies the alias of the table to query (in this case, `User`).
`[`	Indicates the start of a table property reference using a `[` bracket.
`http.querystring.id`	This is a dynamic query based on an HTTP query string parameter of `id`. The primary key value is retrieved from this parameter. The row with this key value is returned from the `User` table if it exists.
`]`	Indicates the end of a table property reference using a `]` bracket.
`.age`	Retrieves the `age` column.
`}`	Indicates the end of the selector using a `}` bracket.

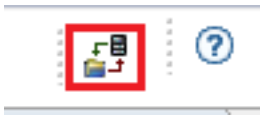
Note Add a **Reflect Message** filter (**Conversion** category) to return a successful HTTP response status of `200`.

5. Set up a path to this policy. In this example, the path is `/kpsGetViaSelector`:



Step 4: Deploy the configuration

When you are finished with your configuration changes, you must deploy them to the API Gateway. To deploy the new configuration, click **Deploy** in the Policy Studio toolbar:



This pushes the configuration to the API Gateway group.

Tip If you deploy an incorrect configuration (for example, specify an incorrect primary key, property type, or name) you can use the `kpsadmin` command to drop the table in storage. For more details, see *Manage KPS using the kpsadmin tool* on page 31.

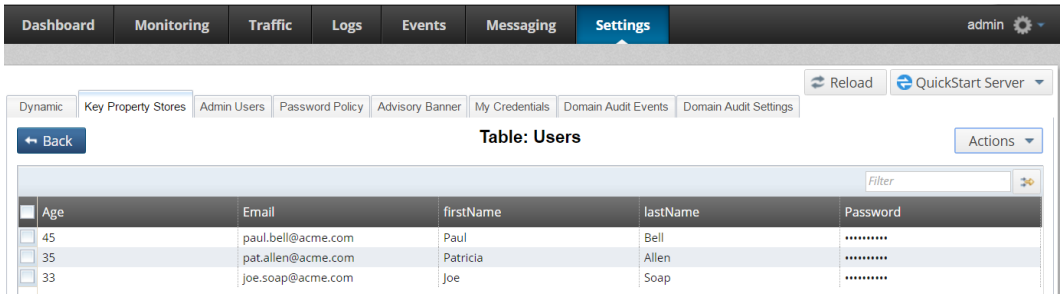
Add KPS data using API Gateway Manager

You can use API Gateway Manager to populate the `User` table with data.

For example, perform the following steps:

1. To access API Gateway Manager in your browser, go to `https://localhost:8090`.
2. Select the **Settings** > **Key Property Stores** > **Samples** > **User** table.
3. To enter new records, select **Actions** > **New Entry** on the right.
4. Click **Save** to save a record.

For example, the table should look as follows:



Access KPS data from a policy

To access KPS data from a policy, go to the following URL in your browser:

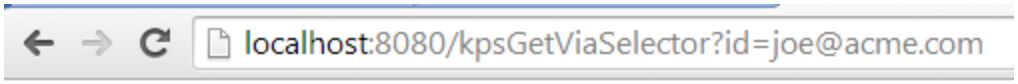
```
http://localhost:8080/kpsGetViaSelector?id=patrica.allen@acme.com
```

This URL specifies the user ID (`email`) as `patrica.allen@acme.com`

You must specify an email that exists in your data. For example:


```
========================
User
===
Email: patrica.allen@acme.com
First Name: Patrica
Last Name: Allen
Age: 35
========================
```

Note If you enter an email that does not exist, you will see `[invalid field]` results. For example:

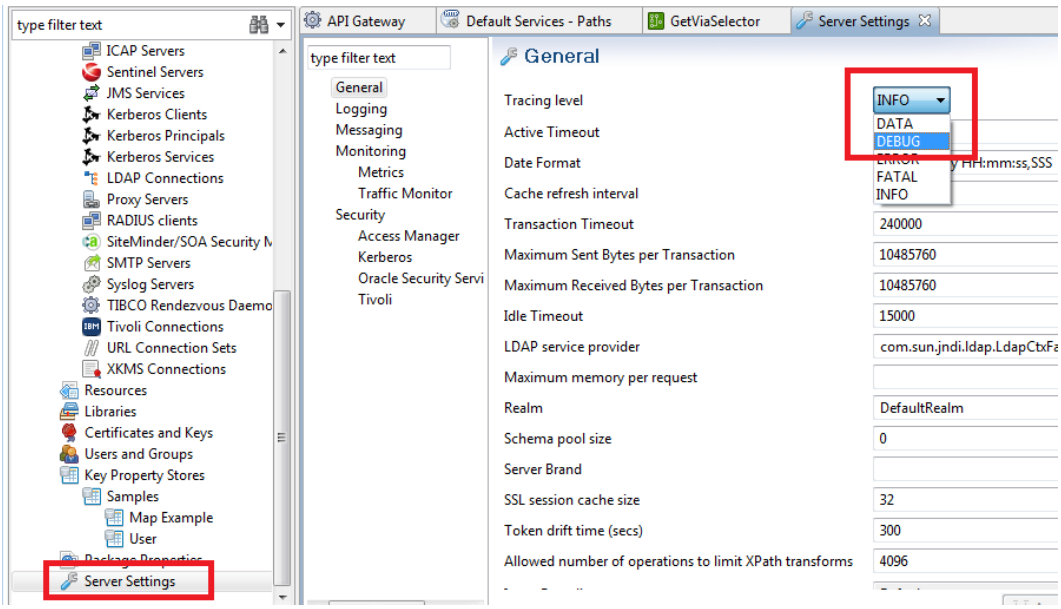



```
========================
User
===
Email: [invalid field]
First Name: [invalid field]
Last Name: [invalid field]
Age: [invalid field]
========================
```

Enable API Gateway tracing

To enable API Gateway debug tracing, perform the following steps in Policy Studio:

1. In the tree on the left, select **Environment Configuration** > **Server Settings** > **General**.
2. Select a **Tracing level** of **DEBUG**.
3. Click **Save**.
4. Click **Deploy**.



Note This setting enables debug tracing for the entire API Gateway, and not just for the KPS.

For more details on API Gateway tracing and logging, see the *API Gateway Administrator Guide*.

Configure KPS in Policy Studio 3

This topic describes how to define general KPS configuration in Policy Studio. For details on data source-specific configuration, see the following topics:

- *Configure Apache Cassandra KPS storage* on page 43
- *Configure database KPS storage* on page 44
- *Configure file-based KPS storage* on page 56

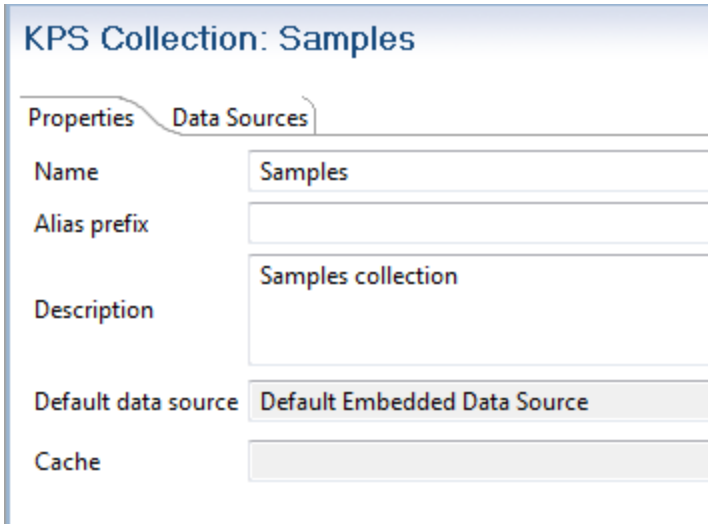
Caution Do not edit the default KPS tables in Policy Studio unless under strict supervision from Axway Support. This includes the **API Server**, **OAuth**, or **API Portal** KPS tables available under **Environment Configuration** > **Key Property Stores**.

Configure a KPS collection

A KPS collection is a set of related KPS tables. To configure a KPS collection, perform the following steps:

1. In the Policy Studio tree, under **Environment Configuration**, right-click **Key Property Stores**, and select **Add Key Property Store**.
2. Specify the following settings in the dialog:
    - **Name**: A collection must have a unique name in the API Gateway group.
    - **Description**: You can provide an optional description.
    - **Alias prefix**: You can also specify an alias prefix, but in normal usage, you can leave this field blank.
    - **Default data source**: A collection has a default data source where all data for all tables in the collection is stored. You can change this data source or assign a different data source to individual tables in the collection.

The following shows a KPS collection created in Policy Studio:



When the collection is created, you can optionally specify a **Cache** for storage and retrieval of selector results. This will improve selector read performance for storage back-ends such as databases.

Note Consider the following before using the **Cache** option.

- Only local caches are supported.
- Do not enable KPS caching for internal tables. For example, API Manager.
- It is not recommended to cache highly dynamic KPS tables without user testing and adjustment of cache (TTL) settings first.
- This cache is intended to increase read performance on KPS stores.
- The cache will be updated when the items in it are expired. If the `Eternal` flag is checked, a restart is required to make the items expired.
- The object gets cached on the first KPS read.
- The cached object does not get automatically invalidated with every update to the KPS.
- The cached object is not removed when deleted from the KPS.
- There is no explicit cache invalidation. You must configure Ehcache based on required use cases.

For more information on local cache, see Global caches.

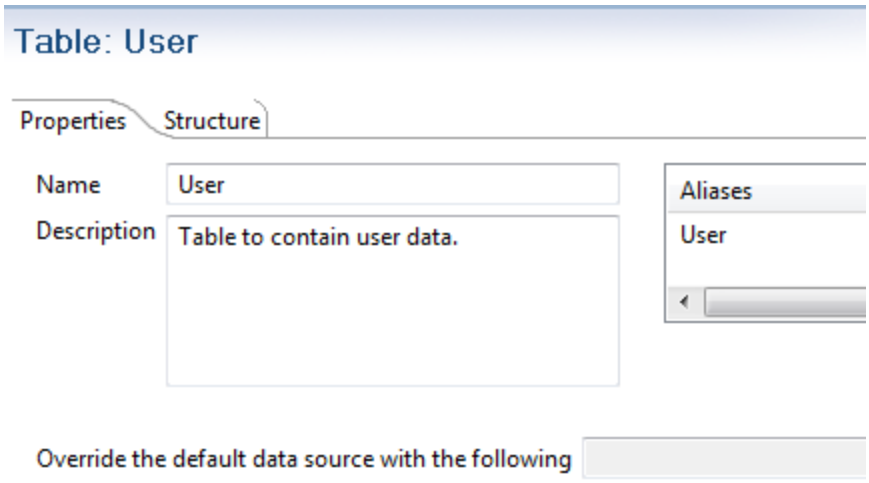
Configure a KPS table

A KPS table is a user-defined table managed by the KPS in the API Gateway. To configure a KPS table, perform the following steps:

1. Right-click a KPS collection in the Policy Studio tree, and select **Add Table**.
2. Specify the following settings in the dialog:

- **Name**: A KPS table must have a unique name in the collection.
- **Description**: You can provide an optional description.
- **Override the default data source with the following**: You can specify a different data source than the collection if required.

3. When the table is created, if required, you can use the **Override the default data source with the following** setting to specify a different data source than the collection:



4. Finally, click the **Structure** tab, and click **Add** to define the structure of the data stored in the table. For details on supported types and keys, see the following:
   - *KPS table structure* on page 26
   - *Query tables using properties and keys* on page 26

KPS aliases

KPS tables are accessed by aliases. A table must have at least one alias. Aliases must be unique in an API Gateway group. You also can use the optional alias prefix for the collection to help ensure that the alias is unique.

The full alias of a table is the *collection alias prefix* and the *table alias* combined. For example, `samples` and `User` gives `samplesUser`. If unspecified, the default value of the alias prefix for the collection is an empty string (for example, `User` only).

KPS data sources

A KPS collection has one active data source associated with it. All tables in the collection use this data source by default. You can configure a table to use a different data source if required (see *Configure a KPS table* on page 24).

KPS table structure

When creating a table, you must define the structure of the data that is stored in that table. This consists of property (column) names and types. You can choose from the following types:

Type	Description

`String`
`Boolean`
`Byte`
`Integer`
`Long`
`Double`
	Java type.

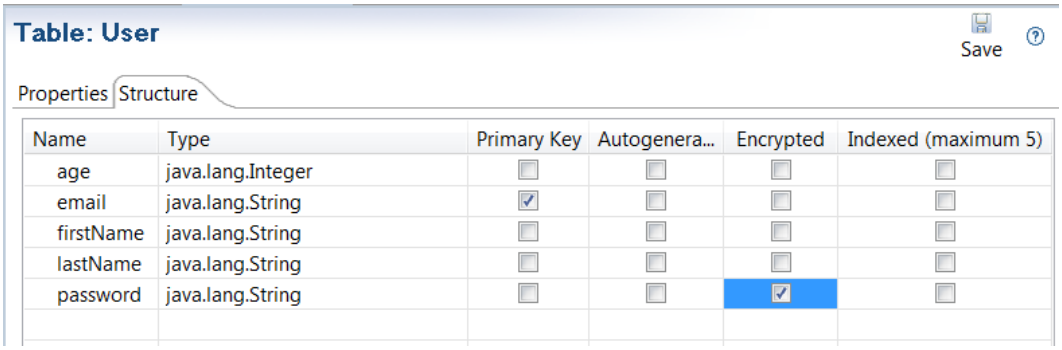
`List`	Java `List` of any one of the above Java types.
`Map`	Java `Map`. The key can be any one of the above Java types. The value can be any one of the above Java types.

Note Do not use just `key` as the property (column) name, because this causes deployment errors.

Query tables using properties and keys

You can directly access records in a table by specifying certain property names and values, without needing to read all the records in the table. You can use only properties of `String`, `Long`, and `Integer` types for this purpose. These properties are known as *indexable properties*.

Indexed properties include primary keys, secondary keys (which are indexed implicitly), and other properties that you explicitly select as **Indexed** in Policy Studio.



Primary key

You can directly access any record using its *primary key*. All records in the table must be accessible using unique primary keys. You must select one **Primary Key** per table in Policy Studio. The specified property must be an indexable property. Primary key values cannot be null.

Secondary key

You can optionally access any record directly using a unique secondary key. The secondary key can be a simple key (for example, `email`) or a composite key (for example, `appId` or `companyId`). The specified properties must be indexable properties. The secondary key (and parts of a composite secondary key) cannot be null. You can specify one secondary key per table. A common use case is to specify an internal unique ID as a primary key, and an external user facing ID as a secondary key. For example, `id` for internal primary key, and `email` for external ID as a secondary key.

Selector access

You can access records in a KPS table using an API Gateway selector. If a secondary key is defined for the table, you must specify all secondary key values in the selector. If no secondary key is defined, you must specify the primary key value instead. In Policy Studio, you can specify a secondary key or a primary key in the **Use the following property name(s) for looking up a table from a selector** field.

For examples of accessing KPS tables using selectors, see the following:

- *Get started with KPS* on page 13
- *Access KPS data using selectors* on page 29
- *Configure database KPS storage* on page 44

Auto-generated properties

In Policy Studio, you can select that `String` fields are **Auto-generated**. When a record is created, a `Java java.util.UUID` is assigned to the field if it is empty.

Note Values for auto-generated fields can be supplied and modified by users. KPS only generates a value at creation time if no value is already present.

Encrypted properties

In Policy Studio, you can select that `String` fields are **Encrypted** in storage. However, fields selected as **Indexed** (including primary and secondary key fields) cannot be encrypted. You can enter values for encrypted fields using the API Gateway Manager or the `kpsadmin` command. These values are forwarded to the API Gateway in the clear using the KPS REST service, and encrypted before being written to storage.

Note The KPS REST service must always run over HTTPS (the default). You must set an encryption passphrase for the API Gateway group, because this is used in the encryption process. For more details, see the *API Gateway Administrator Guide*.

When KPS tables are accessed using API Gateway selectors at runtime, encrypted fields are automatically decrypted. Selectors do not need to be aware that particular fields in a table are encrypted in storage.

When KPS tables are read using the REST API, data is always returned in its encrypted state. Sometimes you may need to view decrypted data to help debug problems on an API Gateway. You can do this using debug mode in `kpsadmin`. This requires you to enter the passphrase for the API Gateway group.

If the in-built KPS encryption mechanism does not suit your needs, you can encrypt and decrypt data outside the KPS. In this case, you should not select properties in KPS tables as encrypted in Policy Studio. Encrypted data must be string-encoded for storage (for example, base64-encoded). Selectors that access the data must decrypt it themselves (for example, using a dedicated decryption filter in Policy Studio).

Access KPS data using selectors

4

Data in KPS tables can be accessed using selectors that execute in policies on the API Gateway at runtime. This topic explains KPS selector syntax and provides some example selectors.

KPS selector syntax

KPS selector syntax is as follows:

```
${kps.alias[key].property}
```

The parts in the selector are described as follows:

Selector part	Description
`${`	Indicates the start of the selector using a `{` bracket.
`kps`	Specifies that selector should query a KPS table.
`.alias`	Specifies the full alias of the KPS table, including the collection alias prefix if any (for example, `User`).
`[`	Indicates the start of a table property reference using a `[` bracket.
key	The key value to query the table (for example, `http.querystring.id`).
`]`	Indicate the end of a table property reference using a `]` bracket.
`.property`	The field to retrieve from the returned row (for example, `age`).
`}`	Indicate the end of the selector using a `}` bracket.

You can also use a composite key, for example:

```
${kps.alias[key1][key2].property}
```

KPS selector examples

For an example of accessing KPS data from a selector using a primary key, see *Get started with KPS* on page 13. For examples of selectors that use both primary and composite keys, see *Configure database KPS storage* on page 44.

The following table shows more examples of KPS selectors:

Selector	Description
`${kps.User [http.querystring.id].firstName}`	• Get row from KPS table with `User` alias • Use key supplied in HTTP query string (`id`) • Return `firstName` field of row
`${kps.User ["kathy.adams@acme.com"].age}`	• Get row from KPS table with `User` alias • Use constant key `"kathy.adams@acme.com"` with quotation marks • Return `age` field of row
`${kps.User [http.querystring.firstName] [http.querystring.lastName].email}`	• Get row from KPS table with `User` alias • Use key supplied in HTTP query string (`firstName` and `lastName`) • Return `email` field of row

For more details on selectors, see the *API Gateway Policy Developer Guide*.

Manage KPS using the kpsadmin tool

5

The `kpsadmin` command-line tool provides KPS managment functions, independent of data source. For example, this includes KPS data backup, restore, encryption, and diagnostics. This topic explains how to use the `kpsadmin` tool in interactive and scriptable command modes.

The `kpsadmin` tool is especially useful in a development environment. In a production environment, you should also use data source-specific tools and administration procedures for data backup, restore, security, optimization, monitoring and so on.

Caution You must use `kpsadmin` operations with caution. Ensure that you have a verified backup before you run destructive operations such as clear, restore, and re-encrypt. You should always try out these options in a development environment first.

For an example of using Cassandra storage, see "Perform essential Cassandra operations" in the *API Gateway Apache Cassandra Administrator Guide*.

Start kpsadmin

From a command prompt, enter `kpsadmin`. For example:

```
INSTALL_DIR/posix/bin/kpsadmin
```

If you do not specify a command operation (for example, `kpsadmin backup` or `restore`), `kpsadmin` enters its default interactive menu mode. For details on available operations in scriptable command mode, see *Run kpsadmin operations in scriptable command mode* on page 37.

In default interactive mode, you are first prompted to enter your admin credentials to authenticate to the Admin Node Manager. These are the credentials that you supplied when installing API Gateway. For more details, see the *API Gateway Installation Guide*.

Start in verbose mode

To run `kpsadmin` in verbose mode, use the `-v` option. `kpsadmin` will then show all REST messages that are exchanged with API Gateway. This is useful for debugging. For example:

```
kpsadmin -v
```

For details on available options, enter `kpsadmin -h`, or see *kpsadmin command options* on page 38.

Select kpsadmin operations in interactive mode

This section describes the `kpsadmin` operations that are available in default interactive mode. When you first select an operation in interactive mode, you must enter the following:

- API Gateway group to use
- KPS Admin API Gateway in that group that handles KPS requests.

Note `kpsadmin` requires you to specify an API Gateway in a group. This is known as the *KPS Admin API Gateway*, which fields KPS requests only. Any API Gateway in the group can be used. For example, you might change this if you want to check that data is available from any API Gateway in the group.

- KPS collection to use in the group
- KPS table to use in the collection

You can change this selection at any time.

KPS table operations

The `kpsadmin` table operations are as follows:

Table operation	Description
Create Row	Create a row in the selected table.
Read Row	Read a row by primary key in the selected table.
Update Row	Update a row in the selected table. The row is specified by primary key.
Delete Row	Delete a row in the selected table. The row is specified by primary key.
List Rows	List all rows in the table.

KPS table administration operations

The `kpsadmin` operations for table administration are as follows:

Table Administration	Description
Clear	Clear all rows in the table.
Backup	Back up the table data. The generated backup UUID is required when restoring the data.

Table Administration	Description
Restore	Restore table data. The table must be empty before you restore.
Re-encrypt	Re-encrypt encrypted data in the table. **Note** This option should only be used if group-level re-encryption fails. For more details, see *Re-encrypt KPS data* on page 41.
Recreate	Recreate a table. This is useful in development if you wish to change the table structure. This procedure involves dropping and recreating the table, so all existing data will be lost. The steps are as follows: 1. Back up (optional). Backup the data if necessary using `kpsadmin`. 2. Deploy the correct configuration. First redeploy the correct configuration using Policy Studio. This may result in some KPS deployment errors. The changes you have made may no longer match the stored data structure. 3. Recreate the table with the correct configuration. Select the Recreate option using `kpsadmin`. 4. Restore (optional) Restore the data using `kpsadmin`. If you have made key or index changes, the data should import directly. If you have made more extensive changes (for example, renaming fields or changing types), you must upgrade the data to match the new table structure.
Table Details	Display information about a table and its properties.

KPS collection administration operations

The `kpsadmin` operations for collection administration are as follows:

Collection Administration	Description
Clear All	Clear all data in all tables in the collection.
Backup All	Back up all data in all tables in the collection.
Restore All	Restore all data in all tables in the collection.
Re-encrypt All	Re-encrypt all data in all tables in the collection. **Note** This option should only be used if group-level re-encryption fails. For more details, see *Re-encrypt KPS data* on page 41.

Collection Administration	Description
Collection Details	Display information about all tables in the collection.

API Gateway group administration operations

The `kpsadmin` operations for API Gateway group administration are as follows:

Collection Administration	Description
Clear All	Clear all data in all collections in the group.
Backup All	Back up all data in all collections in the group.
Restore All	Restore all data in all collections in the group.
Re-encrypt All	Re-encrypt all data in all collections in the group. Use this option when the encryption passphrase has been changed for the API Gateway group. The tables will be offline after a passphrase change. You must use this option to re-encrypt the data. You must enter the old API Gateway passphrase to proceed. Data is re-encrypted using the current API Gateway passphrase. **Note** You must restart the API Gateway instance(s) in the group. For more details, see *Re-encrypt KPS data* on page 41.
Collection Details	Display information about all collections in the group.

Cassandra administration operations

The `kpsadmin` operations for Cassandra administration are as follows:

Cassandra Administration	Description
Show Configuration	Show the current configuration for the KPS storage service (Apache Cassandra).
Run Diagnostic Checks	Run diagnostic checks including HA configuration checks. You must specify if this is a single or multi-datacenter configuration.

General administration operations

The `kpsadmin` operations for general administration are as follows:

General Administration	Description
Change Table	Change the currently selected table.
Change Collection	Change the currently selected collection.
Change Group or API Gateway	Refresh the configuration, and change the currently selected API Gateway group and KPS Admin API Gateway.
Debug Mode	Enable or disable debug mode. To enable, you must enter the API Gateway group passphrase. Encrypted data in KPS tables is then shown in the clear. This can be useful for debugging issues on API Gateway.

Example of switching a data source in interactive mode

This example shows how to switch from Cassandra storage to file storage.

Step 1: Backup collection data using kpsadmin

To copy the current data in the collection to the new data source, back up the collection data using `kpsadmin` option `21) Backup All`.

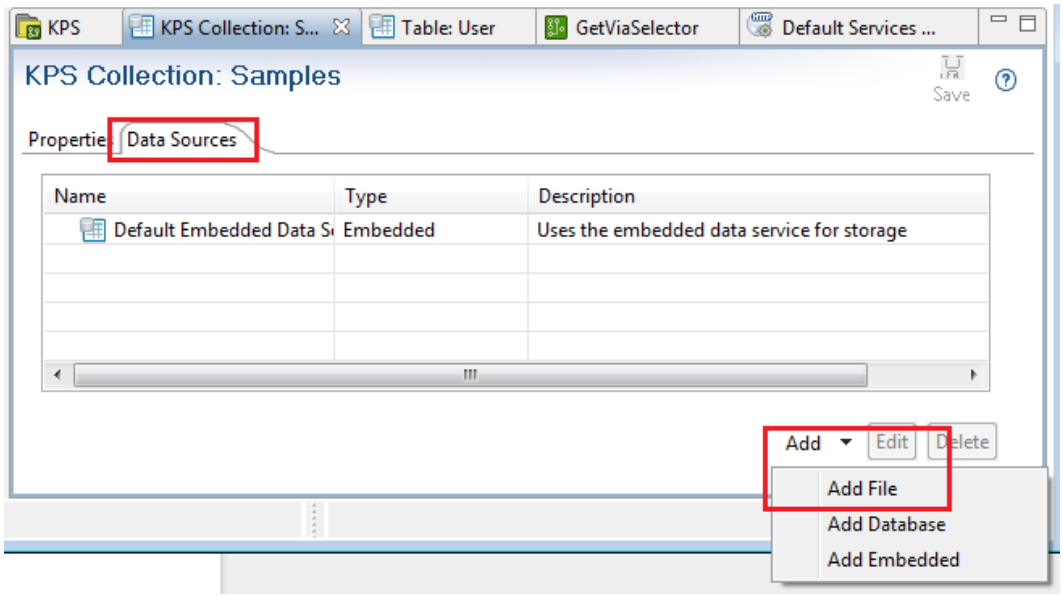
The backup UUID is highlighted in the following example:

```
Select option: 21
Getting Topology...


Select a Group:
1) My Group
Enter selection [1]:


Select an API Server:
1) My Gateway
Enter selection [1]:

Getting model...


Select a Collection:
1) OAuth (internal)
2) Samples
3) API Server (internal)
Enter selection [1]: 2
Backing up all data in all tables in this collection.
Backup uuid is: 892cfd91-4b0a-417d-95f6-6c227100a00b
Are you sure you wish to continue y/n [n]: y
Starting backup to: 892cfd91_4b0a_417d_95f6_6c227100a00b_samples_user_json
Backup done.
```

Step 2: Create a new data source

To create the new data source, perform the following steps:

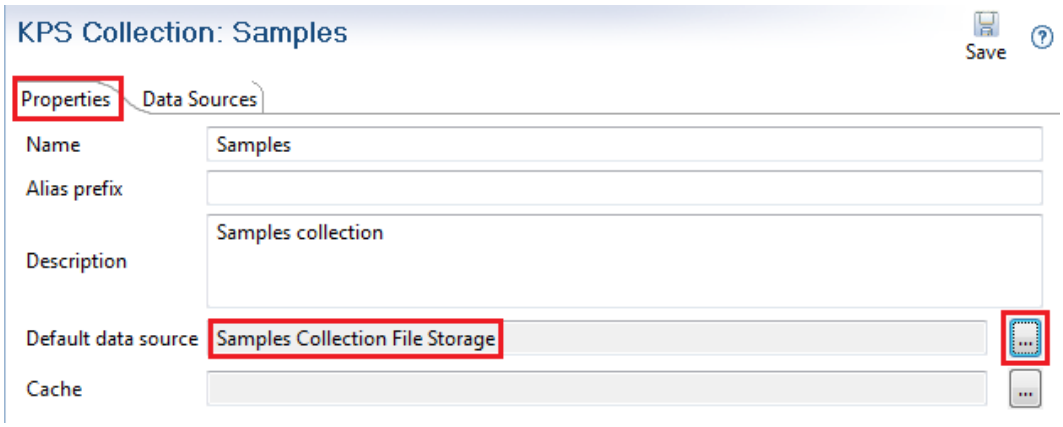
1. In the Policy Studio tree, select **Key Property Stores > Samples**.
2. Select the collection **Data Sources** tab.
3. Click **Add > Add File** at the bottom right.



4. Enter a file data source **Name** and **Description**.
5. Enter a **Directory Path** (for example, `${VINSTDIR/kps/samples`).

Tip You can include `${VINSTDIR}` or `${VDISTDIR}` to indicate the API Gateway instance directory or install directory respectively. Make sure to use `/` on Linux. If the directory does not exist, it is automatically created.

6. Select the collection **Properties** tab.
7. Change the collection **Default data source** to use the new data source:



Step 3: Deploy the configuration

Click the **Deploy** button in the Policy Studio toolbar.

Step 4: Restore collection data using kpsadmin

If you made a backup in step 1, to restore the collection data, perform the following steps:

1. Using `kpsadmin`, select option `22) Restore All`.
2. Enter the backup UUID noted in step 1. For example:

```
Select option: 22
Restoring all data in all tables in this collection.
Enter backup UUID: 892cfd91-4b0a-417d-95f6-6c227100a00b
Are you sure you wish to continue y/n [n]: y
Starting restore from: 892cfd91_4b0a_417d_95f6_6c227100a00b_samples_user_json
.
Restore done.
```

Run kpsadmin operations in scriptable command mode

You can also control `kpsadmin` directly from the command line or from a script by specifying command operations (for example, `kpsadmin backup` or `restore`). If an operation is not specified, `kpsadmin` enters its default interactive menu mode. You must also specify a username/password, and an API Gateway group, KPS collection, or KPS table. For example:

```
./kpsadmin --username admin --password changeme --group "myGroup" --name
"myGateway" backup
```

kpsadmin command operations

The available `kpsadmin` command operations in this mode are:

Operation	Description
`clear`	Clear all data in the specified table, collection, or group.
`backup`	Back up all data in the specified table, collection, or group.
`restore`	Restore all data in the specified table, collection, or group.
`reencrypt`	Re-encrypt all data in the specified table, collection, or group. For more details, see *Re-encrypt KPS data* on page 41.
`details`	Display information about the specified table, collection, or group.
`list_rows`	List all rows in the specified table, collection, or group.
`cassandra_configuration`	Show the current configuration for the KPS storage service (Apache Cassandra).
`diagnostics`	Run diagnostic checks including HA configuration checks. You must specify the `--mdc` option only when this is a multi-datacenter configuration.

Note If this kind of `kpsadmin` command invocation succeeds, `0` is returned. If it fails, `1` is returned. This can be captured on Linux bash shell using `$?` (for example, `echo $?` will work on both platforms). On Linux, `0` means success, `1` means error.

Tip You can specify the username and password on the command line or using a secure script. For details on how to script username and password input for API Gateway scripts, see the `managedomain` command reference in the *API Gateway Administrator Guide*.

kpsadmin command options

The full `kpsadmin` command options are:

Option	Description
`-h, --help`	Show help message and exit.
`-u, --username=USERNAME`	Specify the current Admin Node Manager username.
`-p, --password=PASSWORD`	Specify the current Admin Node Manager password.
`-v, --verbose`	Enable verbose mode for debugging. This shows all REST messages exchanged with API Gateway.
`-g, --group=GROUP`	Specify the API Gateway group to target.
`-n, --name=INSTANCE`	Specify the KPS Admin API Gateway instance that fields all KPS requests for the group.
`-c, --collection=COLLECTION`	Specify the KPS collection to target.
`-t TABLE, --table=TABLE`	Specify the KPS table to target.
`--uuid=UUID`	Specify the UUID required when backing up or restoring data.
`--mdc`	When using the `diagnostics` command, specify that this is a multi-datacenter configuration.

Example kpsadmin scriptable commands

This section shows some example `kpsadmin` operations in scriptable command mode.

Back up and restore

To back up and restore an API Gateway group from a staging environment to a production environment, perform the following steps:

1. Specify the `kpsadmin backup` command:

```
./kpsadmin --username admin --password changeme --group "Staging" --name
"Gateway1" backup
```

2. You must copy the files from the staging `backup` directory to the production `backup` directory and note the UUID. This is output by `kpsadmin` and is also a prefix on the exported filenames.
3. Specify the `kpsadmin clear` command:

```
./kpsadmin --username admin --password changeme --group "Prod" --name
"GateA" clear
```

4. Specify the `kpsadmin restore` command with the UUID noted earlier:

```
./kpsadmin --username admin --password changeme --group "Prod" --name
"GateA" restore --uuid 067e6162-3b6f-4ae2-a171-2470b63dff00
```

Re-encrypt KPS data

After an encryption passphrase change and deployment, you must re-encrypt the KPS data. To re-encrypt at the group level:

```
./kpsadmin --username admin --password changeme --group "Staging" --name
"Gateway1" reencrypt
```

You are prompted to enter the passphrase. For more details, see *Re-encrypt KPS data* on page 41.

Show KPS table details

To show table details:

```
./kpsadmin --username admin --password changeme --group "Staging" --name
"Gateway1" --collection "My Collection" --table "My Table" details
```

Show KPS collection details

To show collection details:

```
./kpsadmin --username admin --password changeme --group "Staging" --name
"Gateway1" --collection "My Collection" details
```

Show Apache Cassandra configuration

To show Cassandra configuration:

```
./kpsadmin --username admin --password changeme --group "Staging" --name
"Gateway1" cassandra_configuration
```

Run diagnostics checks

To output diagnostic information for a group:

```
./kpsadmin --username admin --password changeme --group "Staging" --name
"Gateway1" diagnostics
```

To output diagnostic information for a group in a Cassandra multi-datacenter system:

```
./kpsadmin --username admin --password changeme --group "Staging" --name
"Gateway1" --mdc diagnostics
```

Re-encrypt KPS data

You can use the `kpsadmin` re-encrypt option to re-encrypt previously encrypted KPS data. When you use this option, a backup of the data is first made in the following directory:

```
INSTALL_DIR/apigateway/groups/<group-id>/<instance-id>/conf/kps/backup
```

The data is decrypted with the old encryption passphrase, which you must supply. The data is then re-encrypted with the current encryption passphrase, which API Gateway already knows.

Caution You must use the re-encrypt option *only* when:

- The encryption passphrase has been changed for an API Gateway group configuration.
- This change has been deployed to all API Gateways in the group.
- You see `INFO` messages in all API Gateway trace logs as follows:

```
INFO Loading KPS configuration.
INFO Checking for passphrase changes...
INFO Passphrase change has been detected for the following table(s).
INFO Use kpsadmin to re-encrypt data and passphrase test.
INFO Table(s) will remain in admin mode until this is done.
```

Note You should re-encrypt the data using the group-level `28) Re-encrypt All` interactive option, or the `kpsadmin --group reencrypt` scriptable command. Do not use the table or collection level re-encrypt options. These should only be used if group level encryption fails. You will then need to re-encrypt at collection and/or table level.

After re-encryption, you must restart all API Gateways in the group.

Configure Apache Cassandra KPS storage 6

Apache Cassandra provides a highly available (HA) data storage option for KPS. API Gateway can connect to an external Cassandra database cluster. You must configure three Cassandra nodes to provide a HA data service. A default single Cassandra node, non-HA system typically does not require additional configuration.

Note Custom KPS data defined in Policy Studio supports Cassandra, database, and file data stores. However, API Manager KPS tables (Client Registry and API Catalog) support Cassandra only. Database and file data stores are not supported for API Manager. Three-node Cassandra HA with full consistency is required for API Manager.

For details on installing and configuring Apache Cassandra HA for API Gateway and API Manager, see the *API Gateway Apache Cassandra Administrator Guide*.

Configure database KPS storage 7

KPS data can be stored in a relational database. API Gateway supports the following databases:

- Oracle
- IBM DB2
- Microsoft SQL Server
- MySQL or MariaDB

Note KPS data defined in Policy Studio supports Cassandra, database, and file data stores. API Manager KPS data (Client Registry and API Catalog) supports Cassandra only.

The options for database storage are as follows:

- **Shared database storage**: Data for multiple KPS tables is stored in a single dedicated database table. Data is encoded in JSON before being stored. This approach is very flexible because it allows maps and lists to be stored in tables. It also minimizes administration overhead because only a single database table needs to be created and managed. This is the recommended approach. For more details, see *Shared database storage* on page 44.
- **Per-table database storage** Each KPS table is backed by a single database table. Each property in the KPS table maps to a corresponding column in the database table. This approach allows existing tables to be reused, and allows more precise tuning at database level. However, it also has significant limitations (for example, with supported data types and adding and viewing data). For more details, see *Per-table database storage* on page 48.

Shared database storage

For shared database storage, a single database table is used to store data for multiple KPS tables. This section describes the storage configuration steps.

Step 1: Create a KPS database table

Create a database table called `kps_object` by executing the appropriate SQL script for your database system. SQL scripts are available in the following location:

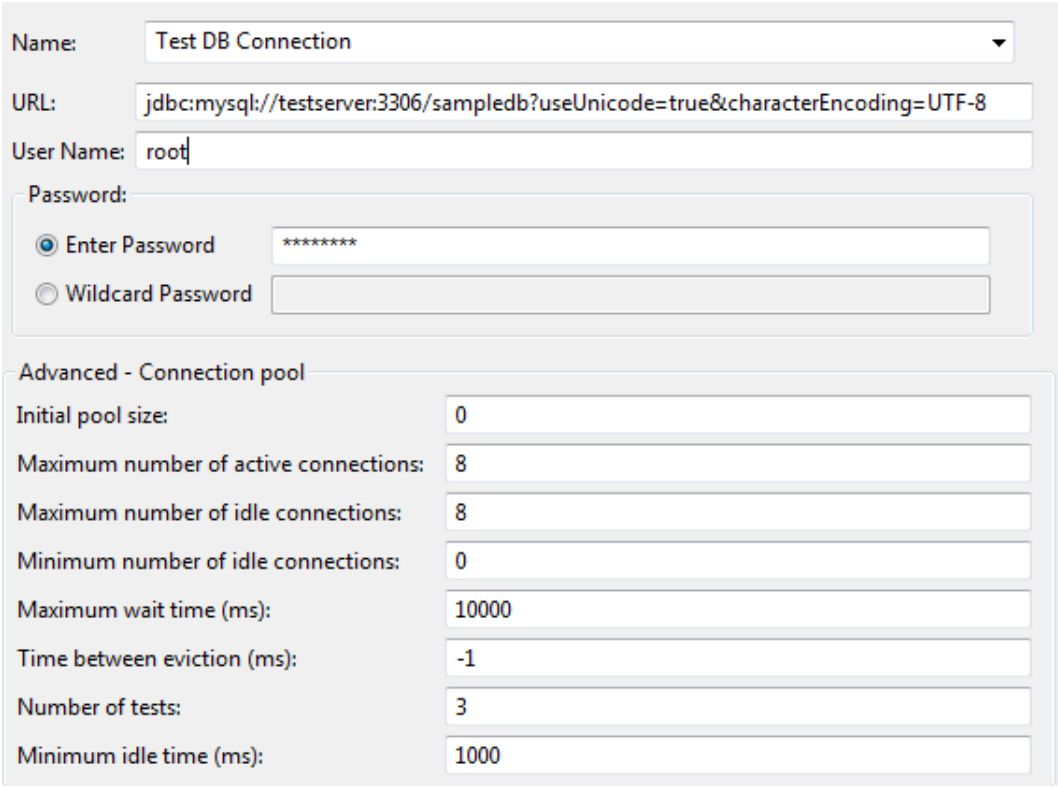
```
INSTALL_DIR/system/conf/sql/DB_NAME/kps.sql
```

Note For Oracle, ensure that the database is created with the AL32UTF8 character set encoding to support UTF-8.

Step 2: Set up an external connection to the database

To access this table from the API Gateway, you must setup an external database connection from the API Gateway. For more details, see the *API Gateway Policy Developer Guide*.

The following shows example database connection settings in Policy Studio:

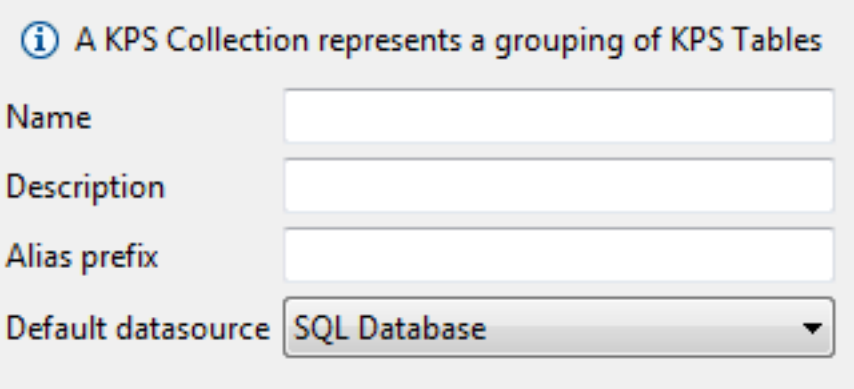


Note For MySQL and MariaDB, the table creation script specifies UTF-8. You must also use the correct JDBC connection URL. Update the connection **URL** field to specify Unicode & UTF-8. For example:

```
jdbc:mysql://testserver:3306/kps?useUnicode=true&characterEncoding=UTF-8
```

Step 3: Use the external connection in a KPS collection

When creating a KPS collection, you can select database storage in the **Default data source** field. The following example shows an SQL database selected:



Alternatively, you can add a database storage option to the collection later. On the **Data Sources** tab, select **Add** > **Add Database**. For an example, see *Step 2: Add the database data source to the KPS collection* on page 49.

For an example with file storage, see *Configure file-based KPS storage* on page 56.

Database storage information

The following describes how KPS data is stored in a database:

- The maximum primary key length in a KPS row is 255 characters
- The maximum KPS table name length is 255 characters
- KPS rows are JSON encoded
- Optimistic locking is used and is enforced using a version column

Increase row size

You can increase the maximum KPS row size by changing the `largeValue` column. For example, to support image icons in MySQL or MariaDB, enter the following command:

```
alter table kps_object modify column largevalue mediumtext;
```

Logging for shared table storage

Shared database storage uses Apache OpenJPA to handle the communication between KPS and the back-end database. You can use OpenJPA logging to view the SQL requests transmitted to the database and their responses. This information can be useful for debugging. You can configure OpenJPA logging using Apache log4j properties.

Enable OpenJPA debug logging

To enable OpenJPA debug logging:

1. Edit the following file:

   ```
   INSTALL_DIR/apigateway/system/lib/log4j2.xml
   ```

2. Add the following settings:

   ```
   Change the level from "error" to "debug" in all the lines shown
   bellow:
   <Root level="info">
   <Logger name="org.apache.openjpa.Tool" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.Runtime" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.Remote" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.DataCache" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.MetaData" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.Enhance" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.Query" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.jdbc.SQL" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.jdbc.SQLDiag" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.jdbc.JDBC" level="error"
   additivity="false">
   <Logger name="org.apache.openjpa.jdbc.Schema" level="error"
   additivity="false">
   ```

3. Restart the API Gateway.
4. Verify that debug statements are written to the log.

Disable OpenJPA debug logging

To disable OpenJPA debug logging:

1. Edit the following file:

   ```
   INSTALL_DIR/apigateway/system/conf/log4j2.xml
   ```

2. Substitute `ERROR` for `DEBUG` in all the `Logger name="org.apache.openjpa.Tool"` entries.
3. Restart the API Gateway.
4. Verify that no debug statements are printed to the log.

For more information on Apache OpenJPA logging, see the Apache documentation.

Per-table database storage

You can use this option to map a KPS table to a single database table. The structure of both tables must match, so a new database table is required for each new KPS table. When the KPS table is queried, an SQL statement is executed to retrieve the correct row from the underlying database table. This SQL statement is provided by the user in Policy Studio.

Tables that use per-table database storage have significant limitations:

- Data cannot be added through KPS, but only directly through the database
- Data cannot be viewed in `kpsadmin` or API Gateway Manager, but can only be read by selectors at runtime
- Tables can only contain simple data types, not maps or lists

KPS tables can be queried using simple or composite keys. This section shows examples of both.

Map a database table using a single key

In this example, a KPS table is accessed using a single key property. This key is used to retrieve the correct row from the database table. This example uses the following `User` table and data created using a MySQL client:

```
CREATE TABLE User (
      email VARCHAR(100),
      password VARCHAR(100),
      firstName VARCHAR(100),
      lastName VARCHAR(100),
      age INT
     );
insert into User (email, password, firstName, lastName, age) values
("ralph.jones@acme.com", "password", "Ralph", "Jones", 30);
insert into User (email, password, firstName, lastName, age) values
("kathy.adams@acme.com", "blah", "Kathy", "Adams", 35);
```

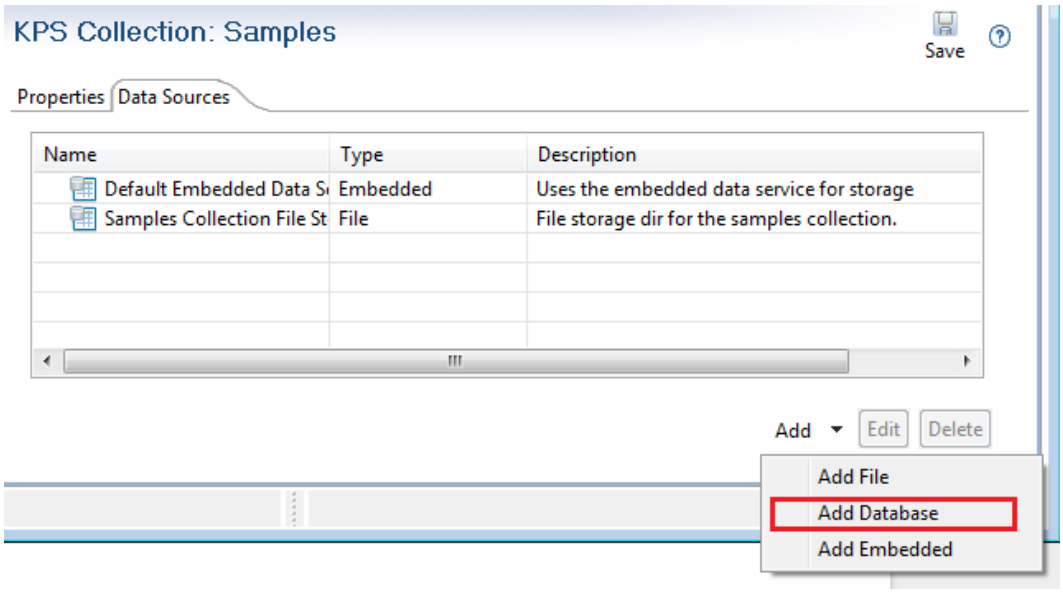
Step 1: Set up an external connection to the database

To access a database from the API Gateway, you must set up an external database connection (see *Step 2: Set up an external connection to the database* on page 45). This example uses the configuration from *Get started with KPS* on page 13.

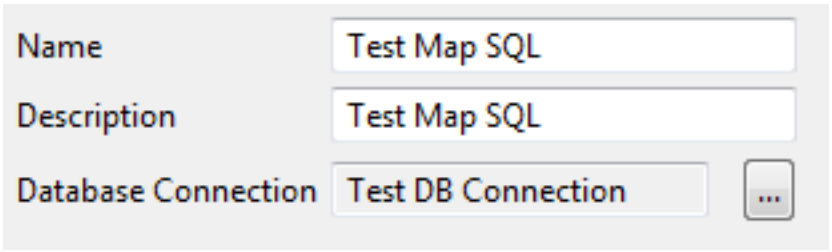
Step 2: Add the database data source to the KPS collection

To add a database data source, perform the following steps:

1. On the KPS collection **Data Sources** tab, select **Add** > **Add Database**.



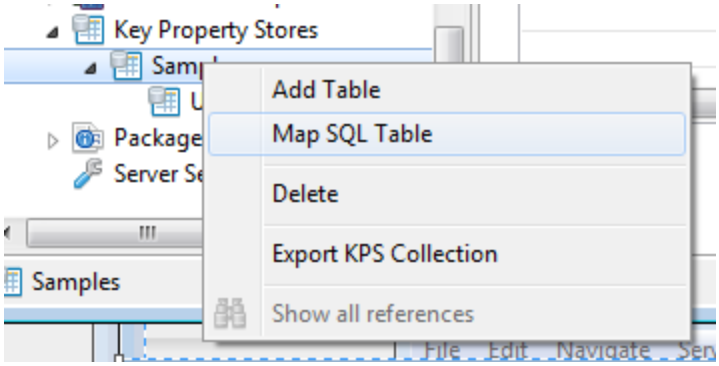
2. Specify the **Database Connection** in the dialog (for example, Test DB Connection):



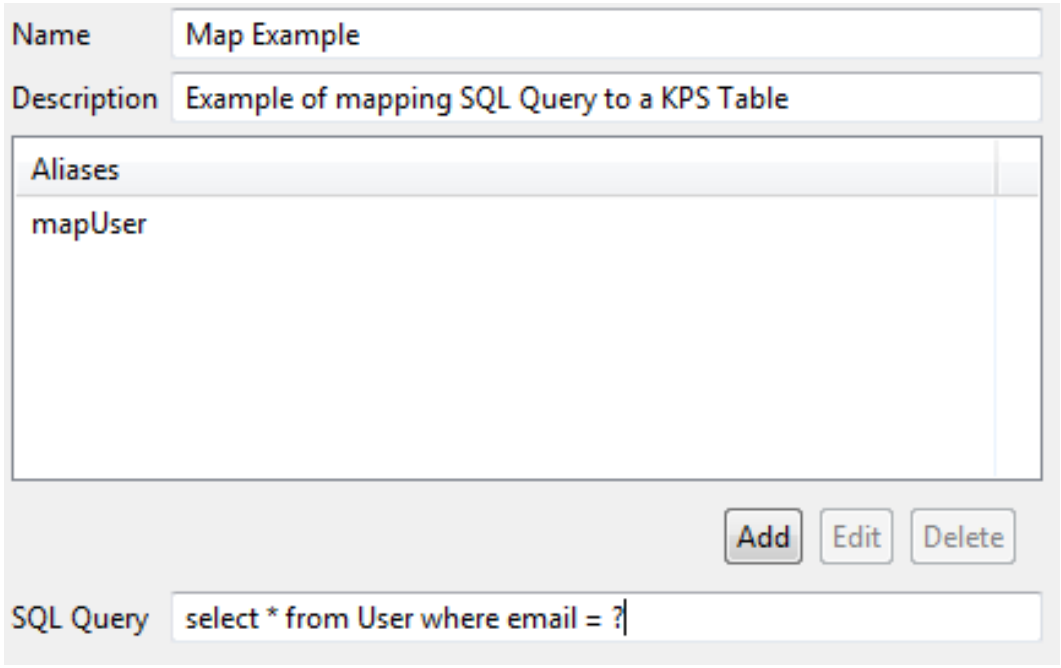
Step 3: Map the SQL table to a KPS table

From an existing KPS collection, perform the following steps:

1. Right-click the KPS collection, and select **Map SQL Table**:



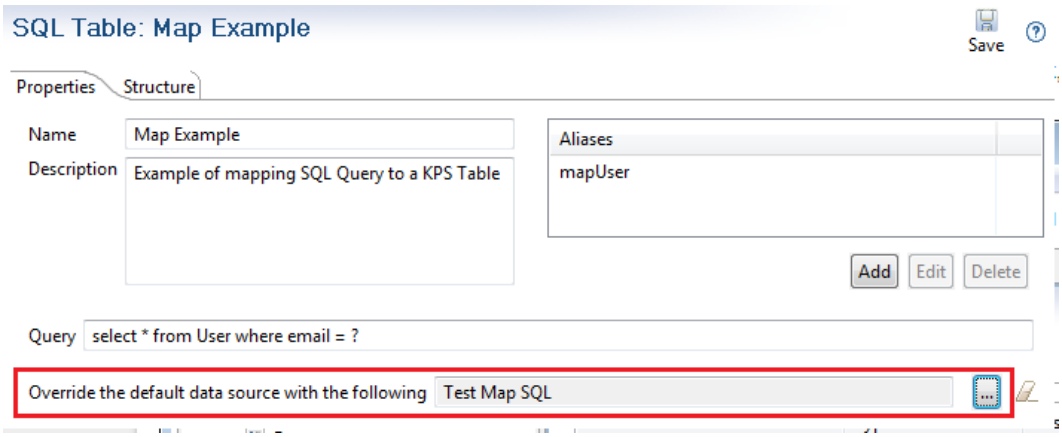
2. Enter an alias in the dialog (for example `mapUser`) :



3. Enter a database-specific JDBC SQL query to retrieve the required data. For example:

```
select * from User where email = ?
```

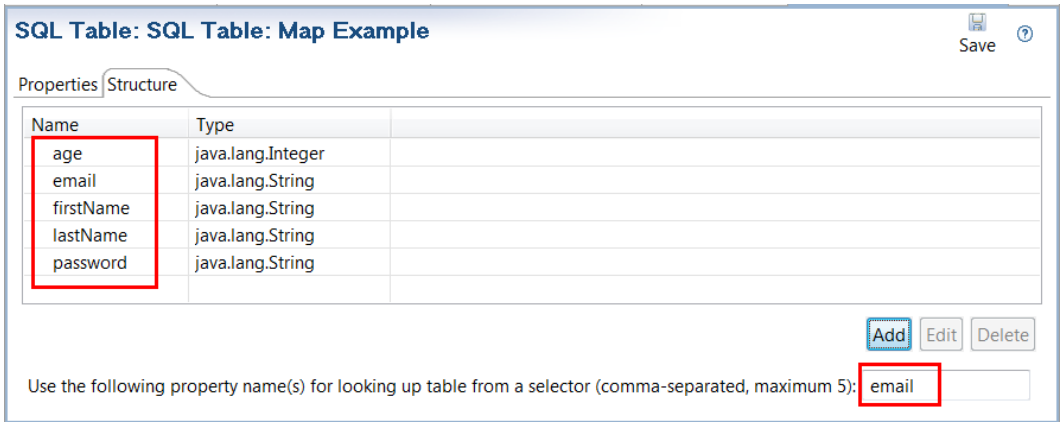
4. On the **Properties** tab of the new KPS table, select the new database data source in **Override the default data source with the following**:



Step 4: Define the KPS table structure

You must define a KPS table structure into which data will be read. You must specify the fields that you expect to read with the SQL query. In this example, all fields in the table are read using an asterisk (*) in the SQL query. This lists all fields, so the order does not matter in this case. However, the names and type must match the result returned by the SQL query.

In this SQL query, `email` is the primary key. You specify `email` as the property to use in corresponding selector queries:



For example, you can use the following selector:

```
${kps.mapUser
["kathy.adams@acme.com"].age}
```

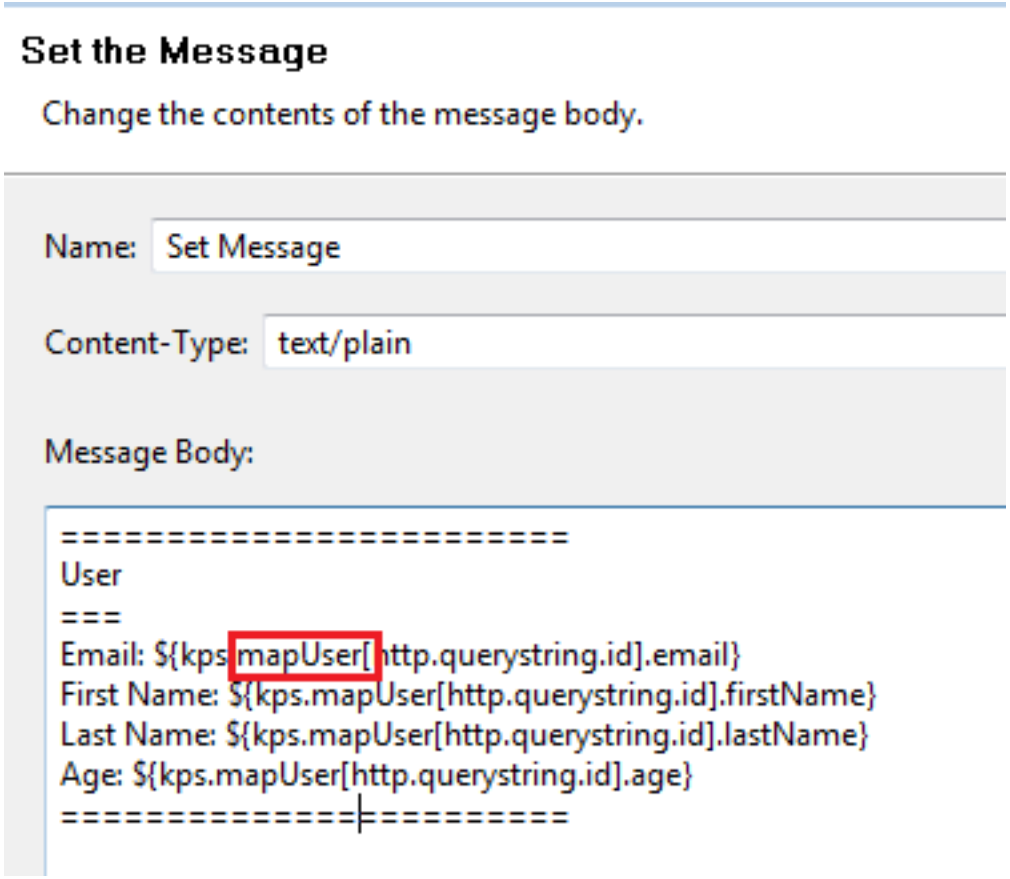
Note This syntax uses ASCII quotation marks (").

This selector generates the following SQL query:

```
select * from User where email =
"kathy.adams@acme.com"
```

Step 5: Define the policy and path

You can reuse the policy and path from *Get started with KPS* on page 13 with one change—use the `mapUser` KPS alias for this new table. For example:



Step 6: Deploy and run

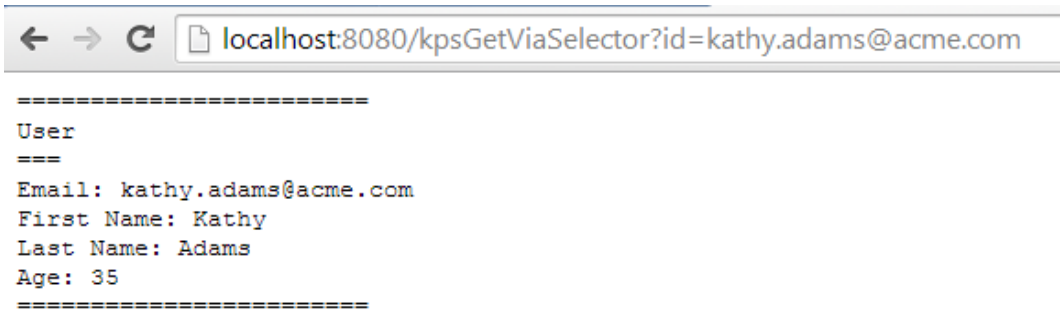
Click **Deploy** in the Policy Studio toolbar.

To run the policy in your browser, go to:

http://localhost:8080/kpsGetViaSelector?id=kathy.adams@acme.com

This URL specifies the user ID (`email`) as kathy.adams@acme.com.

For example, the result is as follows:



How to map a database table using a composite key

This section modifies the example in *Map a database table using a single key* on page 48. This section shows how a table with a composite secondary key can be accessed from a selector. In this version, the secondary key is `{firstName, lastName}`.

Step 1: Modify the KPS table

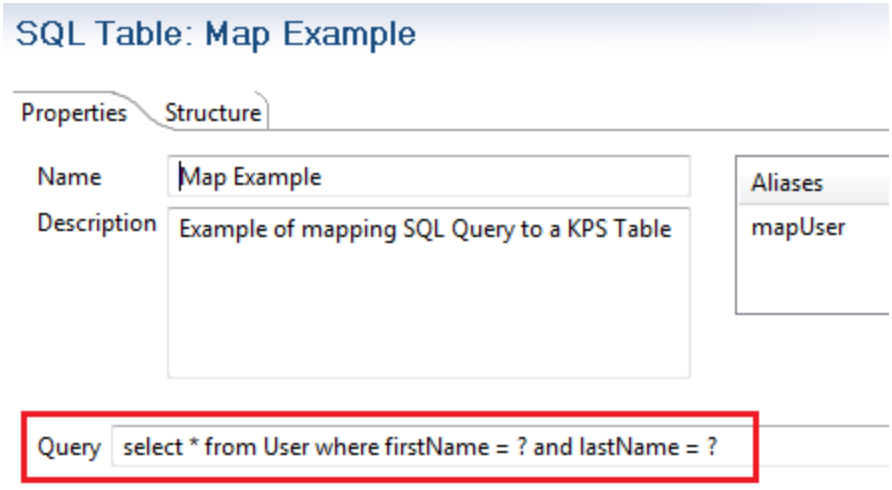
You must update the database-specific JDBC SQL query in the KPS table to retrieve the required data.

To modify the KPS table, perform the following steps:

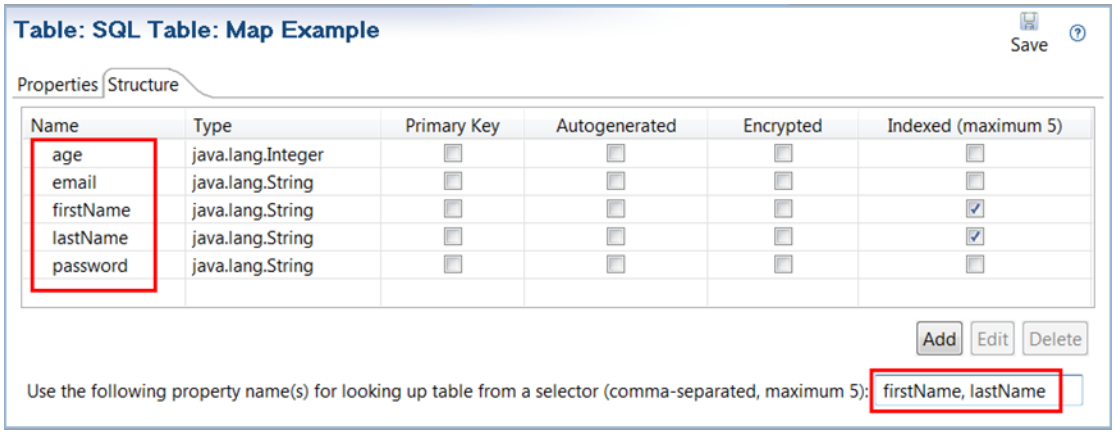
1. On the **Properties** tab in the **Query** field, enter the following:

```
select * from User where
firstName = ? and lastName = ?
```

For example:

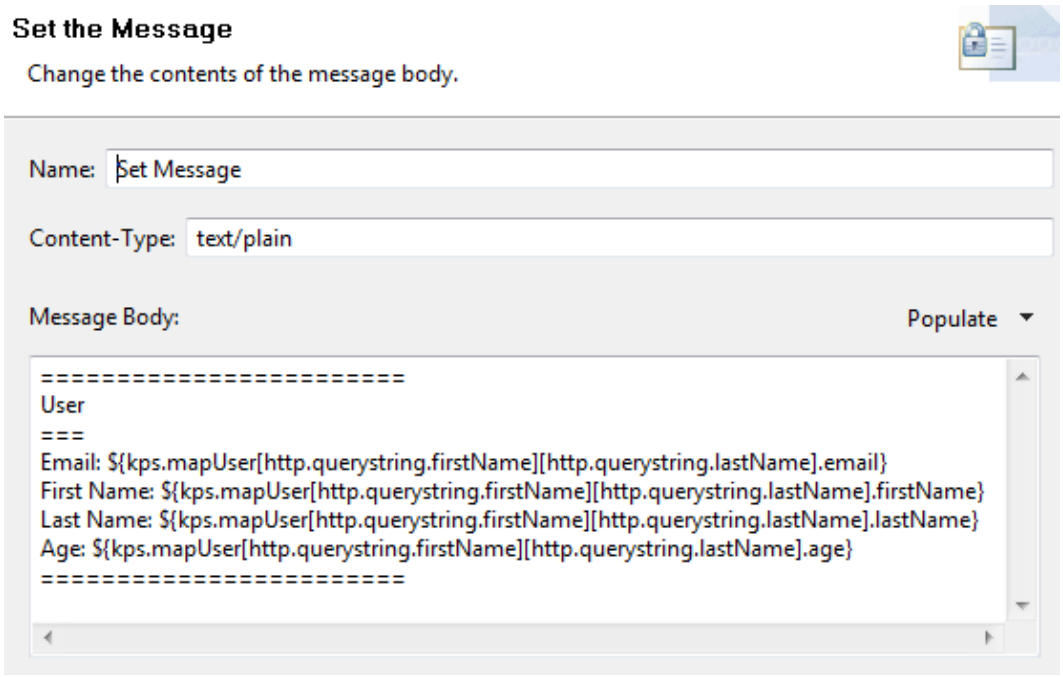


2. On the **Structure** tab, change the selector properties to `firstName,lastName`:



Step 2: Modify the policy

You must update the **Set Message** filter in your policy to use `firstName` and `lastName` parameters. For example:



Enter the following in the **Message Body** field, using a single line for each entry (`Email`, `First Name`, `Last Name`, and `Age`):

```
==========
User
===
Email: ${kps.mapUser[http.querystring.firstName]
```

```
[http.querystring.lastName].email}
First Name: ${kps.mapUser[http.querystring.firstName]
[http.querystring.lastName].firstName}
Last Name: ${kps.mapUser[http.querystring.firstName]
[http.querystring.lastName].lastName}
Age: ${kps.mapUser[http.querystring.firstName]
[http.querystring.lastName].age}
========================
```

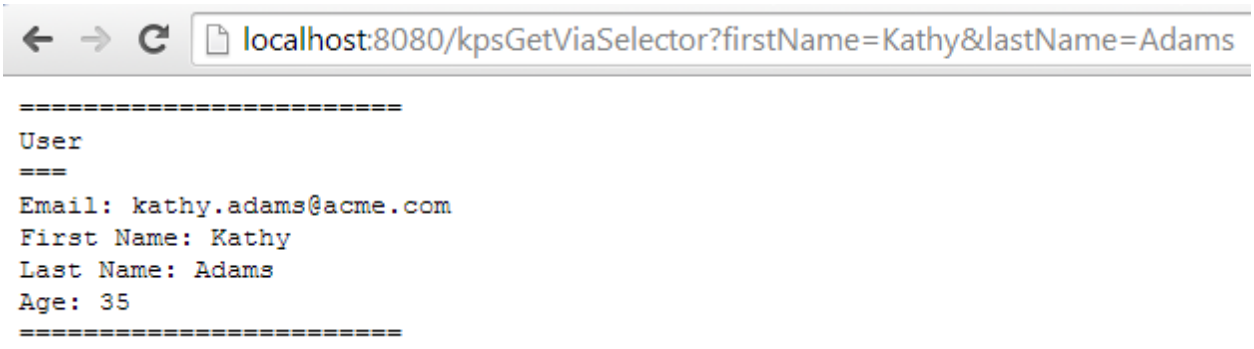
Step 3: Deploy and run

Click **Deploy** in the Policy Studio toolbar.

To run the policy in a browser, go to the following URL:

http://localhost:8080/kpsGetViaSelector?firstName=Kathy&lastName=Adams

For example, the result is as follows:



Configure file-based KPS storage

8

You can store KPS data in a directory on the file system. Each table is stored in a single JSON file. File-based storage is specified at the KPS collection or KPS table level.

Note File-based KPS storage is deprecated and will be removed in a future release.

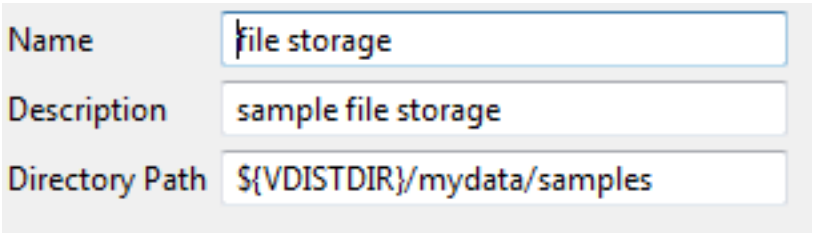
File-based KPS storage is most suited to single API Gateway deployments. In a multi-API Gateway scenario, file replication or a shared disk is required to ensure that all API Gateways use the same data.

KPS data defined in Policy Studio supports Cassandra, database, and file data stores. API Manager. KPS data (Client Registry and API Catalog) supports Cassandra only.

File-based KPS tables are read and cached by API Gateways when they start up. If data is modified, all API Gateways must be restarted to pick up the changes.

Configure a file-based KPS collection

You can configure a file-based KPS data source when creating a KPS collection, or add one later on the **Data Source** tab. For example, the following settings are available when editing file-based collection:



These settings are described as follows:

Name	Description
Name	Collection-unique data source name.
Description	Optional description.

Name	Description
Directory Path	The directory name where table data for the collection is stored. If the directory does not exist, it is automatically created. If this directory is not specified, the directory path defaults to `${VINSTDIR}/conf/kps`. The path can include `VDISTDIR` or `VINSTDIR` variables. These are resolved to the API Gateway instance and installation directories. For example, `${VDISTDIR}/mydata/samples`. Remember to use the correct path separator (`/` on Linux).

Further information

See also the following:

- *Configure Apache Cassandra KPS storage* on page 43
- *Configure database KPS storage* on page 44

Use the KPS scripting API 9

The KPS scripting API facilitates create, read, update, delete (CRUD) operations to KPS tables from your policies. You can use this API via a custom **Scripting Language** filter in Policy Studio.

Prerequisites

This topic assumes that you are already familiar with using a KPS, and that you know how to define KPS tables and read data from KPS tables using selectors. For more information, see the following topics:

- *Configure KPS in Policy Studio* on page 23
- *Access KPS data using selectors* on page 29

Usage guidelines

When using Apache Cassandra as a KPS:

- It is recommended that you set up Cassandra as detailed in the *API Gateway Apache Cassandra Administrator Guide*.
- For a Cassandra-backed KPS, this API allows you to set a Time to Live (TTL) value for each record. The TTL specifies a time period after which a record expires and becomes unavailable from the table. If you set a TTL, you must specify all fields that are not auto-generated when creating or updating records.

  You should also set the Cassandra `gc_grace_seconds` value to a value larger than the TTL value. This ensures that data is purged sooner than the default (10 days).
- If you plan to update the KPS table frequently, it is recommended that you configure the Cassandra data source with a **Read Consistency Level** and **Write Consistency Level** of `ONE`.

When configuring KPS tables to use a local Ehcache:

- In this case the local Ehcache Time to Live (TTL) or Time to Idle (TTI) settings might conflict with the TTL parameters specified to KPS. You must test your use case carefully and you should never use an Ehcache TTL/TTI value that is bigger then the KPS table TTL value.
- When configuring the cache in Policy Studio, you must ensure that you do not select the **Eternal** option for a local cache, as this results in the specified TTL/TTI values being ignored.
- Since the caches are local to each API Gateway, you might have eventual inconsistencies between the cache and the KPS, for any record updates or deletes. The time during which

eventual consistencies for a record happens is defined by the cache TTL.

- If you plan to mostly read from the KPS table, it is recommended that you use a local cache.

Method descriptions

The API is defined in the java package `com.vordel.kps.Table`. It has the following methods.

Create record in table

```
Map<String, Object> createRecord(String tableAlias, Map<String, Object>
record, int ttl) throws ObjectExists, ObjectNotFound
```

Parameters

`tableAlias` – alias of table to use.

`record` – record to create, conforms to table definition in KPS (excluding auto-generated fields).

`ttl` - record time to live, in seconds. If 0, TTL is ignored.

Returns

Returns the newly created record (with auto-generated fields completed if necessary).

Throws

`ObjectExists` – if a record with the same primary key already exists.

Update record in table

```
Map<String, Object> updateRecord(String tableAlias, Map<String, Object>
record, int ttl) throws ObjectExists, ObjectNotFound
```

Parameters

`tableAlias` – alias of table to use.

`record` – record to update, conforms to table definition in KPS (excluding auto-generated fields, primary key field is mandatory).

`ttl` – record time to live, in seconds. If 0, TTL is ignored.

Returns

Returns the updated record.

Throws

`ObjectNotFound` – if no record with the given primary key exists.

Read record from table

```
Map<String, Object> readRecord(String tableAlias, Object primaryKey)
throws ObjectNotFound
```

Parameters

`tableAlias` – alias of table to use.

`primaryKey` – primary key of the record to read.

Returns

Returns the record with given primary key.

Throws

`ObjectNotFound` – if no record with the given primary key exists.

Delete record from table

```
deleteRecord(String tableAlias, Object primaryKey) throws ObjectNotFound
```

Parameters

`tableAlias` – alias of table to use.

`primaryKey` – primary key of the record to delete.

Returns

Nothing.

Throws

`ObjectNotFound` – if no record with the given primary key exists.

Example code

The following Jython code examples show how you can use the KPS scripting API in a **Scripting Language** filter. For more information on using this filter, see "Scripting language filter" in the *API Gateway Policy Developer Guide*.

Create a new record

```
from java.util import HashMap
from com.vordel.kps import Table, ObjectExists

def invoke(msg):

    record = HashMap()
    # If the primary key is auto-gererated then, the next line should be deleted
    record.put("Primary key Name", <Primary Key Value>)
    # Repeat next line for each of the non-auto-generated properties properties,
note that if TTL is non-zero then, all non-auto-generated properties must be set
    record.put("Property Name", <Property Value>)

    try:
        record = Table.createRecord("table Alias", record, <Integer setting ttl in
seconds>)

    except ObjectExists:
        #Handle case when an object with the given primary key already exists
```

Update a record

```
from java.util import HashMap
from com.vordel.kps import Table, ObjectNotFound

def invoke(msg):
```

```
    record = HashMap()
    record.put("Primary key Name", <Primary Key Value of record to update>)
    # Repeat next line for each of the properties to update, note that if TTL is
non-zero then, all properties must be set
    record.put("Property Name", <Property Value>)

    try:
        record = Table.updateRecord("table Alias", record, <Integer setting ttl in
seconds>)
        # Note that, the updated record TTL has been reset to value used in the last
parameter

    except ObjectNotFound:
        # Handle case when an object with the given primary key does not exists
```

Extend TTL for a record

```
from java.util import HashMap
from com.vordel.kps import Table, ObjectNotFound

def invoke(msg):

    record = HashMap()
    record.put("Primary key Name", <Primary Key Value of record to update>)
    # Repeat next line for each of the properties to update, note that if TTL is
non-zero then, all properties must be set
    record.put("Property Name", <Property Value>)

    try:
        record = Table.readRecord("table Alias", <Primary Key Value>)
        Table.updateRecord("table Alias", record, <Integer setting ttl in seconds>)
        # The record TTL is now extend to the new value

    except ObjectNotFound:
        # Handle case when an object with the given primary key does not exists
```

Read a record

```
from com.vordel.kps import Table, ObjectNotFound

def invoke(msg):

    try:
        record = Table.readRecord("table Alias", <Primary Key Value for record to
```

```
read>)

    except ObjectNotFound:
        # Handle case when an object with the given primary key does not exists
```

Delete a record

```
from com.vordel.kps import Table, ObjectNotFound

def invoke(msg):

    try:
        record = Table.deleteRecord("table Alias", <<Primary Key Value for record to
delete>)

    except ObjectNotFound:
        # Handle case when an object with the given primary key does not exists
```

Appendix A: KPS FAQ

This appendix answers the frequently asked questions on KPS and API Gateway, KPS storage, and Apache Cassandra.

KPS and API Gateway

This section includes frequently asked questions on KPS and API Gateway:

What is KPS used for in API Gateway?

In addition to use with your API Gateway policies, KPS is used as an option for OAuth token storage and Client Application Registry. KPS storage in Apache Cassandra is required for API Manager and API Portal.

What is KPS not suitable for?

KPS is not suitable for complicated data models, ad hoc queries, or where full ACID transaction support is required. KPS does not enforce referential integrity.

What are the transaction semantics of KPS?

Individual KPS operations are atomic (A), isolated (I), and durable (D). Consistency (C) depends on the data storage mechanism chosen and the number of API Gateways in a group.

With file storage, data is consistent in a single API Gateway. With supported database storage, data is consistent. Cassandra storage allows consistency levels to be set per KPS table. KPS does not provide transactions across multiple operations. You cannot issue a set of KPS operations and roll them back.

What is the KPS collection alias prefix for?

This provides an optional namespace for a KPS collection to help ensure that tables in the collection have a unique alias. In most cases, you can leave this prefix empty.

How do I change the API Gateway group passphrase?

To change the group passphrase, perform the following steps:

1. Change the group passphrase in Policy Studio. For details, see the API Gateway *Administrator Guide*.
2. In `kpsadmin`, select the `Collection Administration`, `Re-encrypt All` option to re-encrypt the data in each collection.
3. You will be asked to enter the old API Gateway passphrase. This passphrase is used to decrypt the data. The data is then re-encrypted with the current API Gateway passphrase.
4. Restart the API Gateway instance(s) in the group.

For more details, see *Manage KPS using the kpsadmin tool* on page 31.

KPS storage options

This section includes frequently asked questions on KPS storage mechanisms:

Is Apache Cassandra storage required? Can you use file or database storage?

Apache Cassandra is the default out-of-the-box storage, but is not a requirement for API Gateway. You can switch to database or file storage. For more details, see *How do you switch storage for a KPS collection?* on page 65

Note KPS data defined in Policy Studio supports Cassandra, database, and file data stores. API Manager KPS data (Client Registry and API Catalog) supports Cassandra only. File-based KPS storage is deprecated and will be removed in a future release.

How do you switch storage for a KPS collection?

The general steps to switch storage for a KPS collection are as follows:

1. Back up the data for a collection using `kpsadmin`.
2. Add and configure a new data source (file, database, or Cassandra) using Policy Studio.
3. Deploy the configuration in Policy Studio.
4. Restore the data using `kpsadmin`.

For more details, see the example in *Manage KPS using the kpsadmin tool* on page 31.

Why use database storage?

You may already have a relational database system and expertise that you wish to use. For more details, see *Configure database KPS storage* on page 44.

Why use file storage?

Note File-based KPS storage is deprecated and will be removed in a future release.

File storage is very easy to use. Data is written to very simple JSON encoded files. File storage is suitable for single API Gateway use in development and production. If you have more than one API Gateway sharing the files, you must restart these API Gateways after a KPS update. It is only suitable here for rarely changing, read-mostly data. For more details, see *Configure file-based KPS storage* on page 56.

When can you use kpsadmin? When should you use storage-specific tools?

You can use `kpsadmin` anytime. However, it is especially useful for development use. In production, you should also use storage-specific tools and procedures (for example, for data backup). For more details on `kpsadmin`, see *Manage KPS using the kpsadmin tool* on page 31.

Apache Cassandra

This section includes frequently asked questions on the Apache Cassandra database:

Why use Cassandra as a KPS storage option?

Cassandra is a key-value store. Cassandra has a non-restrictive Apache 2.0 license. It provides high availability, and has an active community.

What versions of Cassandra are supported by API Gateway?

For details on supported Cassandra versions, see *Install an Apache Cassandra database* in the *API Gateway Installation Guide*.

What does all host polls marked down mean?

This means that the client cannot connect to the Cassandra server. For details on how to resolve this issue, see *Troubleshoot KPS error messages* on page 68.

Appendix B: Troubleshoot KPS error messages

This appendix describes KPS error messages, and explains how to resolve them.

All platforms

This section explains how to troubleshoot KPS errors on all platforms:

All host polls marked down

This error means that the API Gateway client cannot connect to the Cassandra server.

To resolve this issue, perform the following steps:

- Check that all ports and addresses are correct in `cassandra.yaml` to verify that the endpoints are what you expect.
- Enable Cassandra debug logging.
- Contact your network administrator

For more details on Cassandra configuration, see *Install an Apache Cassandra database* in the *API Gateway Installation Guide*.

Further information

This topic shows where to find more details on KPS-related information:

Feature	Documentation
KPS REST API	The KPS REST API is documented in the API Gateway REST API documentation, available from the Axway Documentation portal at https://docs.axway.com, and also included in your API Gateway installation at `INSTALL_DIR/apigateway/samples/swagger`.
Apache Cassandra	For details on how to install and configure Apache Cassandra, see the *API Gateway Installation Guide*. For more details on Apache Cassandra, see the following: • http://cassandra.apache.org/ • https://docs.datastax.com/en/cassandra/2.2/
API Gateway database connections	For details on how to configure database connections, see the *API Gateway Policy Developer Guide*.
API Gateway caches	For details on how to configure local caches, see the *API Gateway Policy Developer Guide*.

Glossary

Apache Cassandra

An open source distributed database, designed to provide high availability by distributing data across multiple servers.

KPS

Key Property Store. Data management component in the API Gateway.

KPS alias

An alternative name for a KPS table that is unique in an API Gateway group.

KPS collection

A collection of related KPS tables.

KPS table

A user defined table that is managed by the KPS in the API Gateway.

Selector

A special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime based on metadata values (for example, from a KPS, message attribute, or environment variable).