# Table of Contents

Version: 1.0 / 2022-03-31

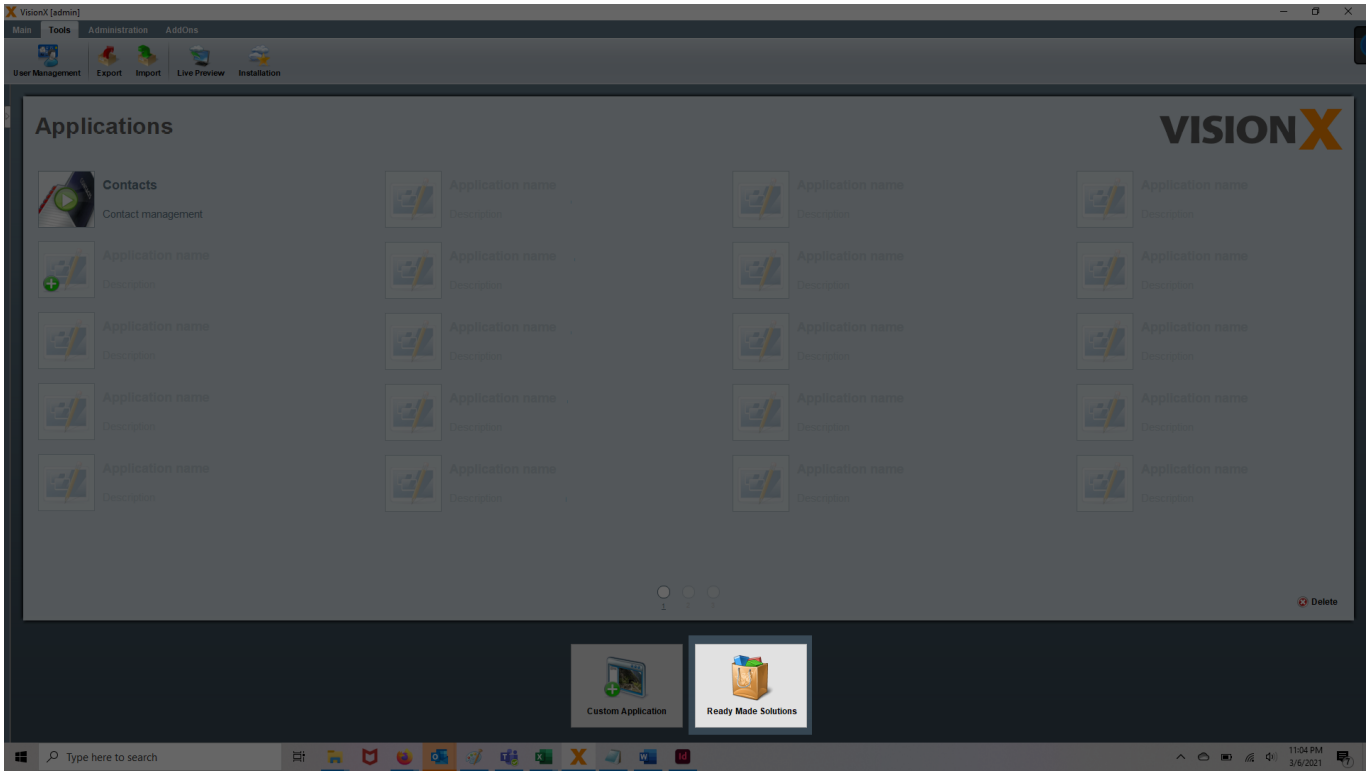
# Introduction

REST services is the best way to connect different applications, systems and services. It is the standard for interfaces between applications. You could also say everything is an API.

The VisionX REST Add-on supports more than 50 REST services out of the box, and over 2,000 apps through Zapier. Our powerful CRUD REST support makes working with REST services as easy as working with database tables, with just a few clicks.

# Installation

Before you can use the REST services in VisionX, the REST add-on has to be installed. On the VisionX start screen, click on the “Ready Made Solutions” store at the bottom of the screen.




The click on “Add-ons” on the left menu.

And click the “+” icon next to the “REST Services” add-on.

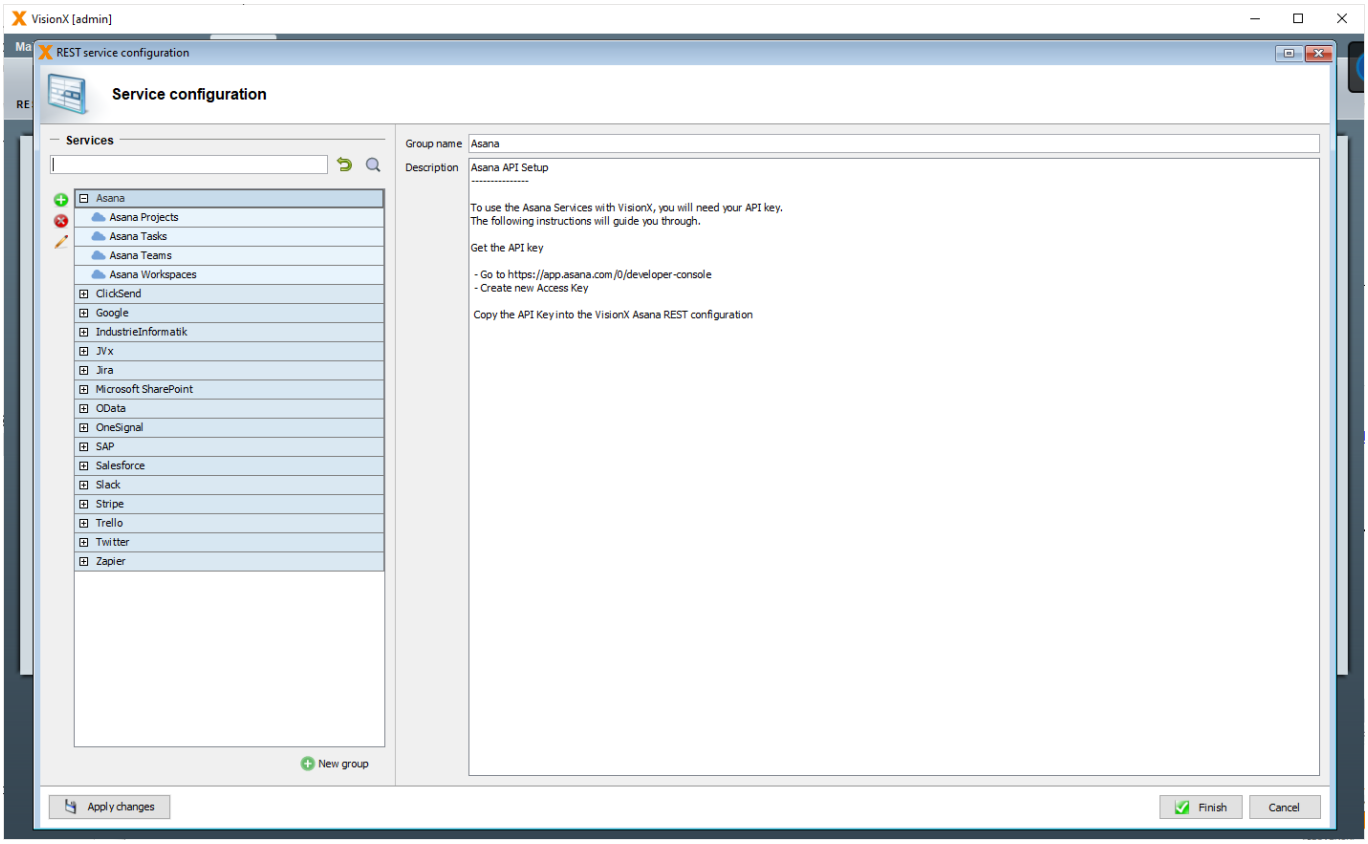
VisionX must now be restarted for the add-on to be activated.

# Configuring Services

*Note: Every REST API is a little different. While we made it as easy as possible to configure connections, the exact steps and requirements will vary. Please consult the API documentation for the requested resource for details on URLs, authentication, and headers.*

On the VisionX start screen, click “Add Ons” on the menu at the top of the screen and then “REST Services”.

This takes us to the REST service configuration screen. Here you can see the preconfigured services for SaaS products such as Asana, Salesforce, Slack, Cronet and others. Please refer to the “Description” section of each preconfigured service for more details on how to use the connection.




To configure a new service, first click on “New Group” at the bottom of the screen and enter a name for the group.

After creating a group, click on the “+” icon at the top left of the screen.

*Please note that for configuring a new service, both a new group and a new service are required.*

You can now enter a name and description for the service, and select from the following options:

Type:

- **Tabular Data:** this is the most common type - our CRUD REST service that is used to connect classic table-based REST services as easily as database tables. A separate REST service ist used for each CRUD function: Fetch (=Read, a typical HTTP GET request), Insert (=Create or an HTTP POST request), Update (=Update, or an HTTP PUT request) and Delete (Delete or an HTTP delete request).
- **Action Call:** used to make REST services available as actions in VisionX applications. For example, an action can be defined for a button to insert a new record using a POST request. See the section „VisionX REST Action Calls“ below for more details.

Binary Response: Used for REST services that return files or binary data (e.g. for downloading files from One Drive).

Active: Displays the service’s status (active/inactive). Inactive services cannot be used in VisionX applications.

On the tabset in the middle of the screen, you can see the various available operations: FETCH, INSERT, UPDATE, DELETE, METADATA and AUTHORIZATION.

You can select the relevant method from the dropdown, and enter the URL for the requested resource.

If required, authorization credentials can be entered on the “AUTHORIZATION” tab and enter the appropriate credentials.

First, select the authorization method from the dropdown. Details on the authorization method can be found in the API documentation of the selected REST service.

In our example we use basic authorization with the username “admin” and password “admin”.

You can find more details on authorization methods in the chapter - AUTHORIZATION

After entering the credentials we switch back to the “FETCH” tab.

If any headers are required, you can add them in the table below by clicking on the ”+“ icon. Please check the API documentation for the requested resource for header information.

# Fetching test data

For this example, we are going to use a demo resource that is available at the following URL with username: admin and password: admin

[Heroes Demo Resource]
(https://cloud.sibvisions.com/heroes/services/rest/Heroes/HeroesWorkScreen/data/heroes)

After selecting “Get” as the method from the dropdown on the “Fetch” tab and entering the URL and authorization details, click on the “Test FETCH” button on the top right.

You will now see the results in the “Test result” table on the bottom right of the screen.

In this case, it is a simple table with two columns: ID and Name.

The formatted results are displayed on the “Data (formatted)” tab:

The original JSON data is shown on the “Data (1:1)” tab:

In the column named “Use as column”, we can now select the columns we want to use in the VisionX table. In this case, since we only have a total of two columns, we select them both. Once selected, they appear in the “Columns” table at the bottom of the screen:

Path: The result of the REST call may not be returned at the highest level of the JSON response. For example, a result may be returned below the „details” level of the hierarchy, in which case the path has to be set to „details“ for the results to be recongized accurately.

In the “Columns” table, we can now adjust the data type, length, precision, scale or format if necessary. We can also set one of the columns to be a primary key. These are relevant for the unique identification of data records, and are required for “Update” or “Delete” operations.

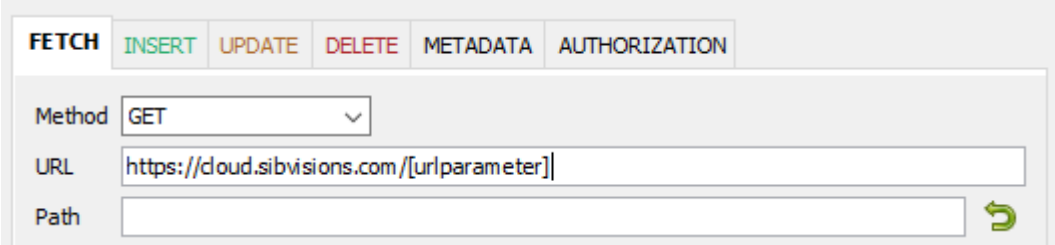
Our table is now ready to use in VisionX. Click “Finish” to complete the configuration.

# Using Parameters

Parameters can be used for any of the configuration data instead of hardcoded values. To create a parameter, simply enter its name ins square brackets.

For example, the authorization credentials can be entered as parameters:

We can also use a parameter for parts of a URL:




Once a parameter has been defined, it will show up in the “Parameters” section on the right side of the screen:

For each parameter, we can decide if it is a config parameter or if it is entered during runtime, by selecting from the dropdown.

- **Config Parameters:** these are parameters that are defined when the REST service is added to the application. They do not change when the application is running.

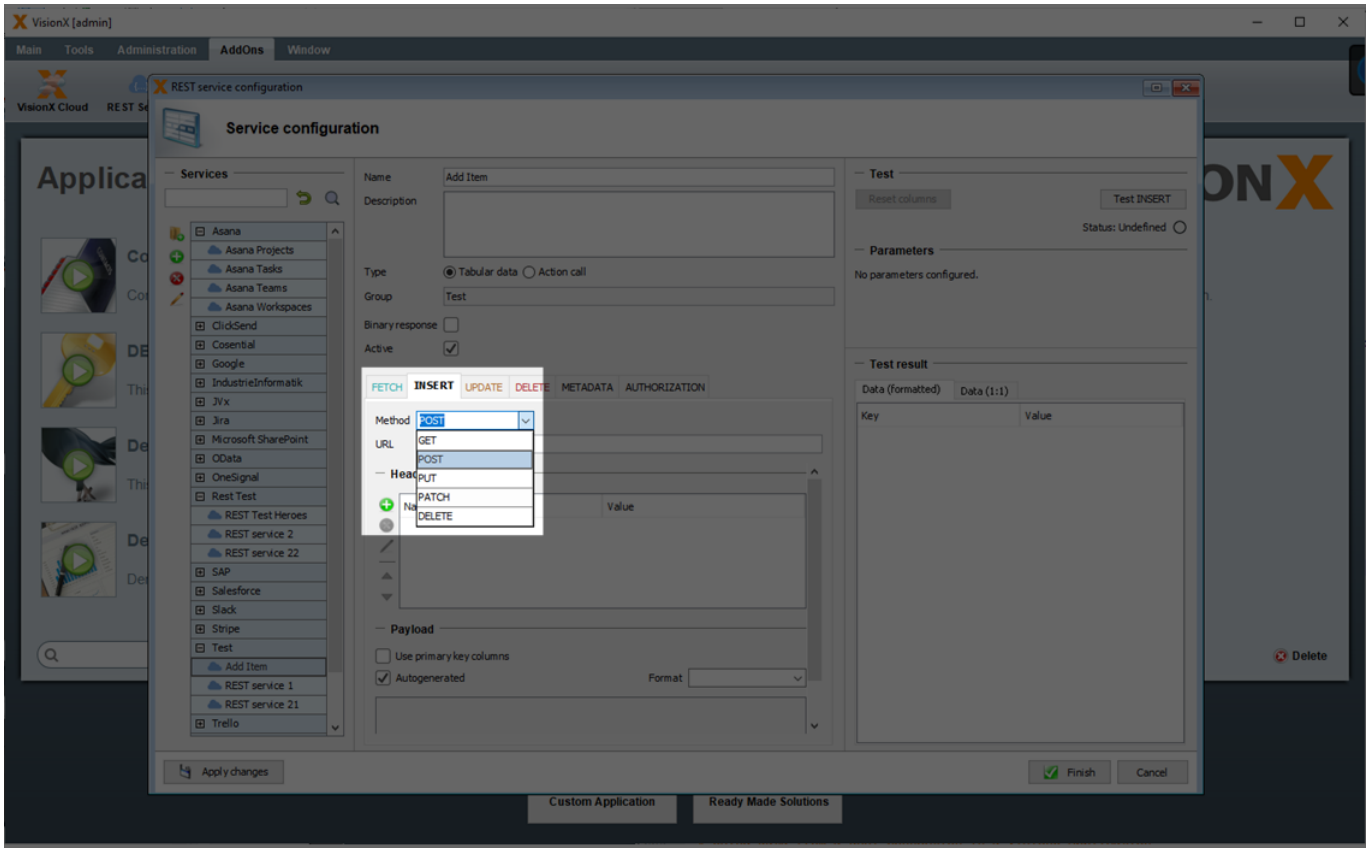
Examples of common config parameters are usernames, passwords, and URLs. See also Using Data in an Application

- **Runtime Parameters:** these are parameters that are used to retrieve values when the application is running. When can use them to modify REST calls based on values in the application data.

# INSERT

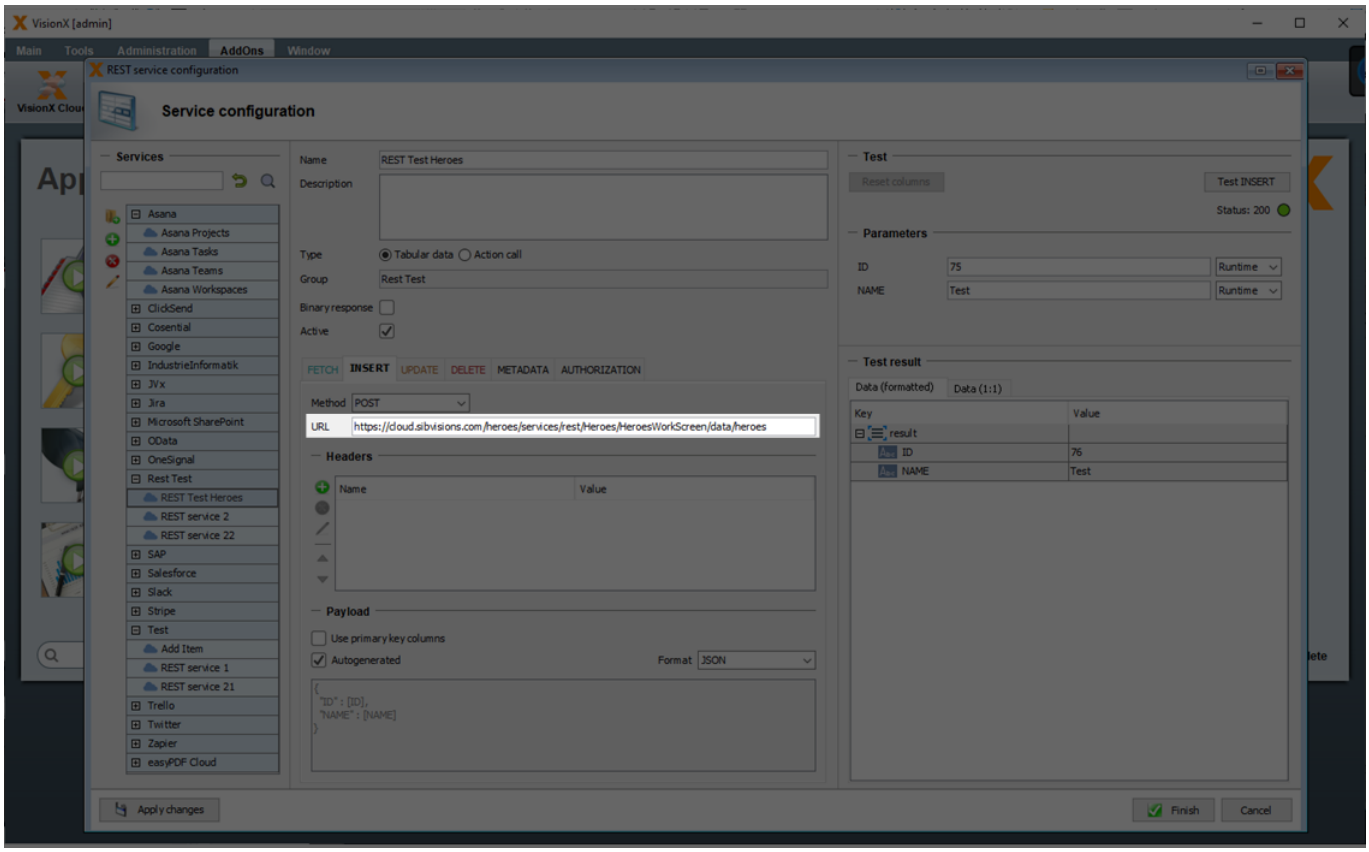
The “INSERT” request is used to create a new database record; it is usually done using the HTML POST method.

To create a new record, click on the “INSERT” tab and select the “POST” method from the dropdown menu.




We now have to enter the URL. We are going to use the demo resource again:

https://cloud.sibvisions.com/heroes/services/rest/Heroes/HeroesWorkScreen/data/heroes



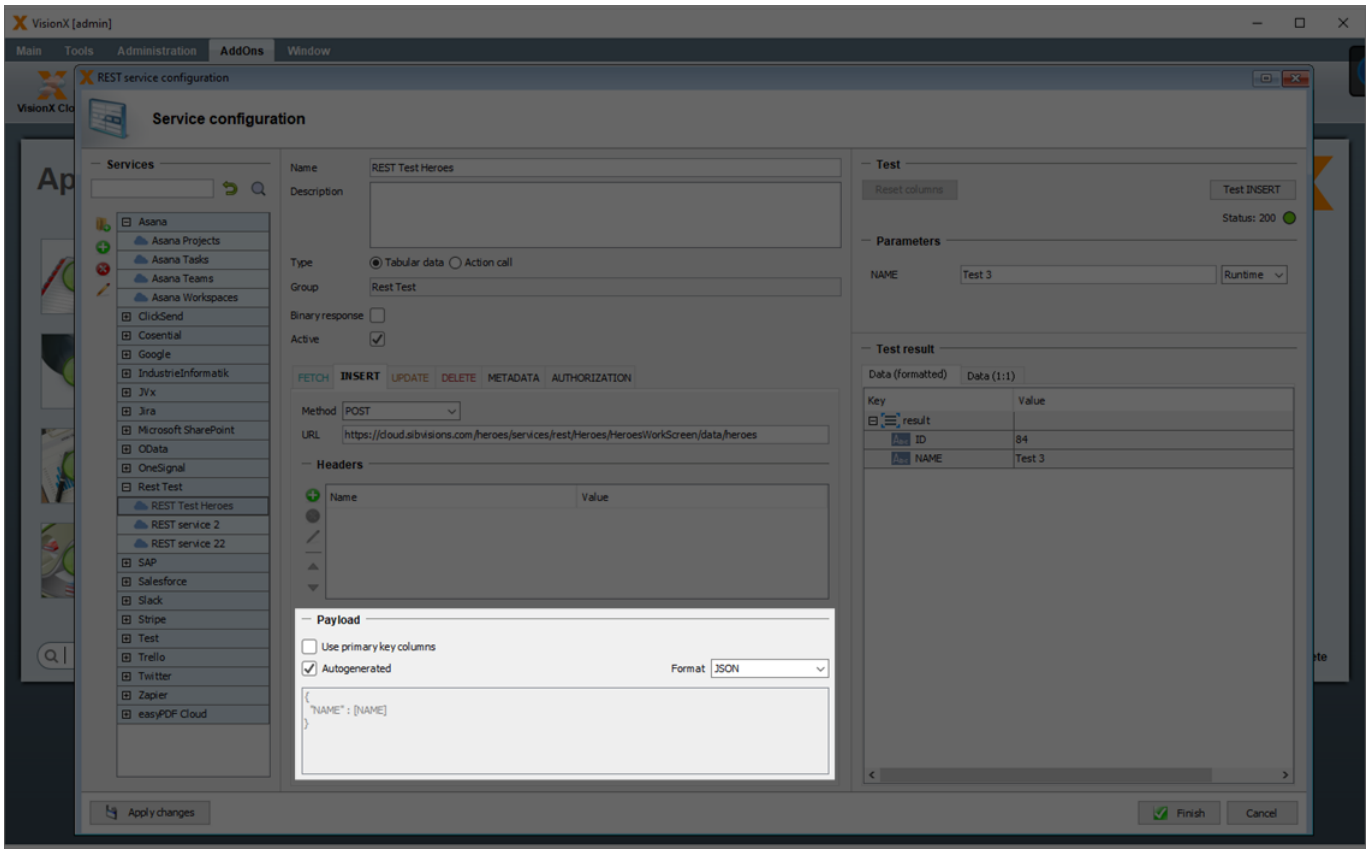

Next, we have to define the parameters for the data that we want to insert. This is done in the “Payload” section at the bottom of the screen.

The payload is autogenerated by default, using all of the columns previously defined. *Note: Primary key column are only included in the autogenerated payload if we check the “Use primary key columns” box.*

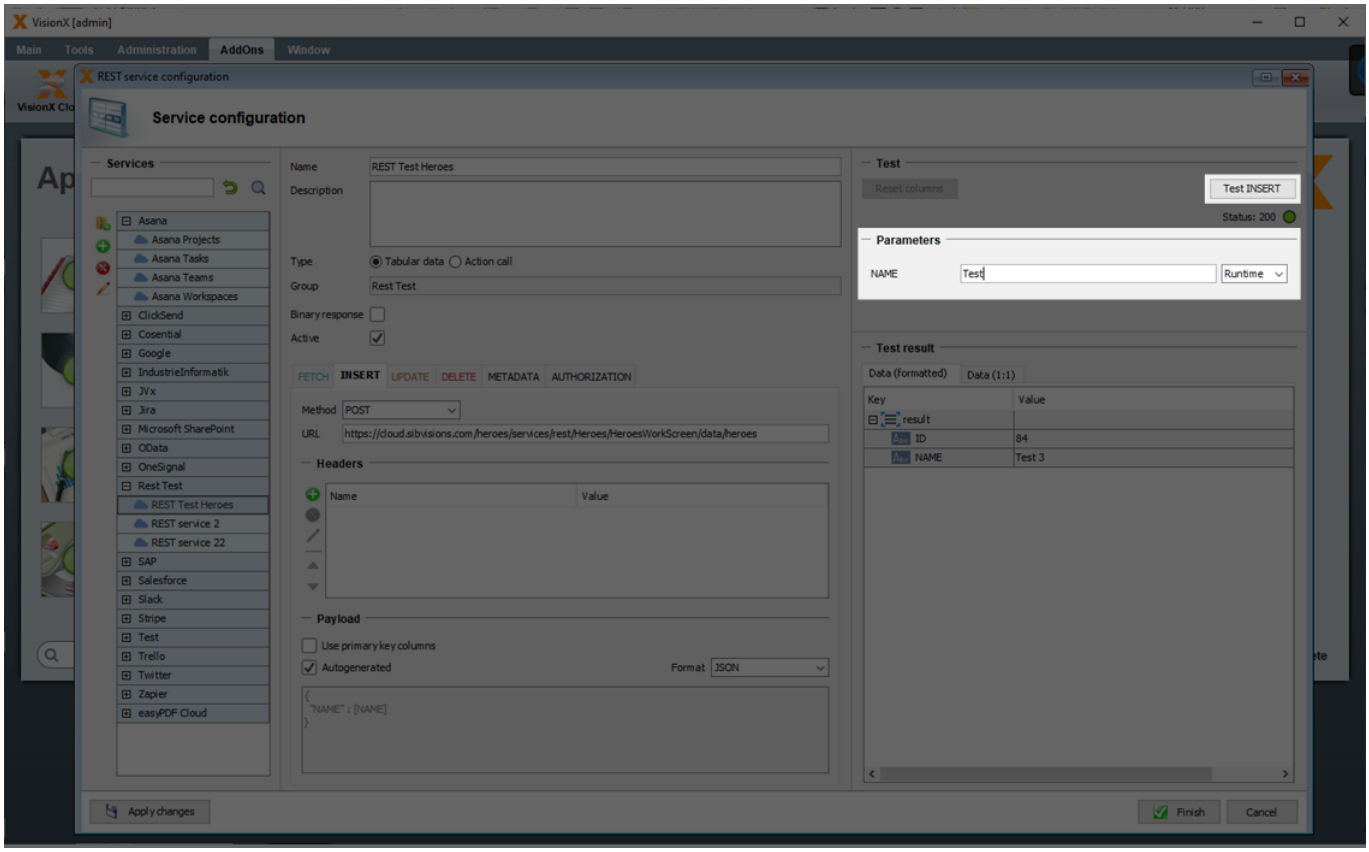
In our example, we are only going to include one column: NAME.

If we want to change the payload definition, we can simply uncheck the “Autogenerated” checkbox and edit as necessary. For this example, we are not going to make any changes to the payload, so we will just keep the autogenerated text.

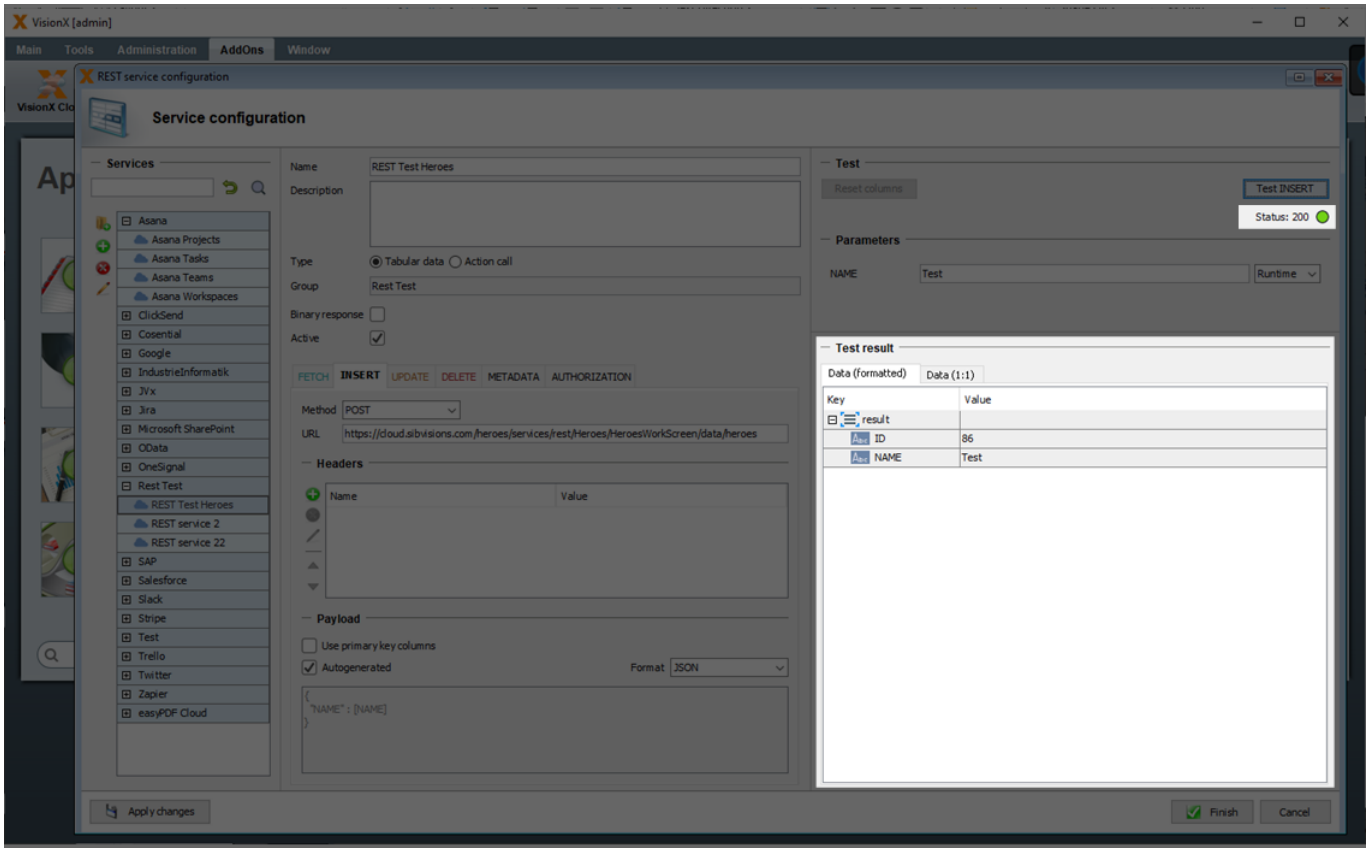
The payload format can be selected from the dropdown menu on the right. The following formats are available: Text, Javascript, JSON, HTML, XML, Binary, and Multipart.




We can now test inserting a record by entering a Name under “Parameters” on the right side of the screen, and then pressing the “Test INSERT” button.




The 200 status code means that the record was successfully inserted. We can see the result in the “Test result” window.

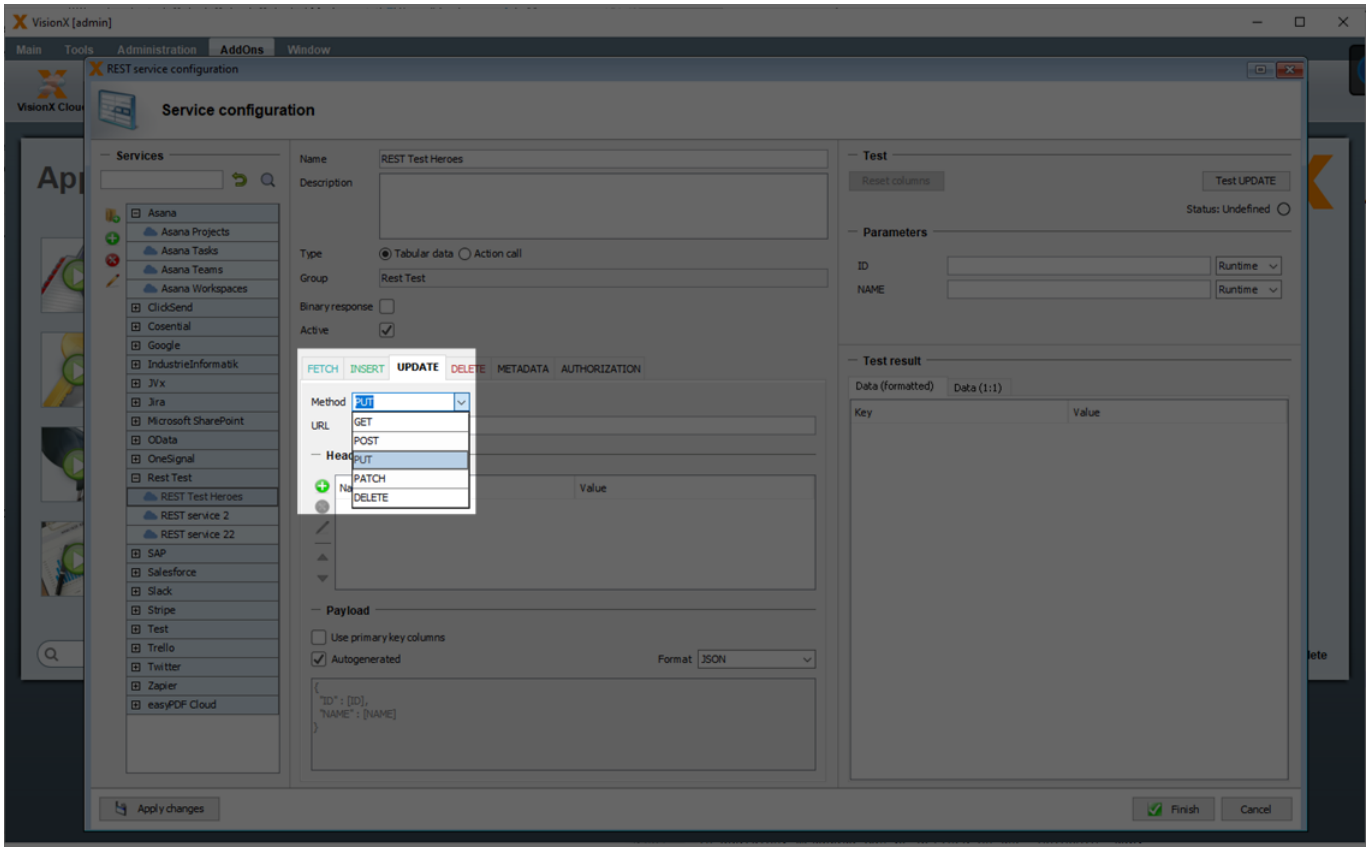



*Note: REST services vary in their requirements for specifying columns when sending an “INSERT” request. Please refer to each service's REST documentation for details.*

# UPDATE

The “UPDATE” request is used to update an existing record and is usually done using the HTML PUT method.

To update a record, click on the “UPDATE” tab and select the “POST” method from the dropdown menu.

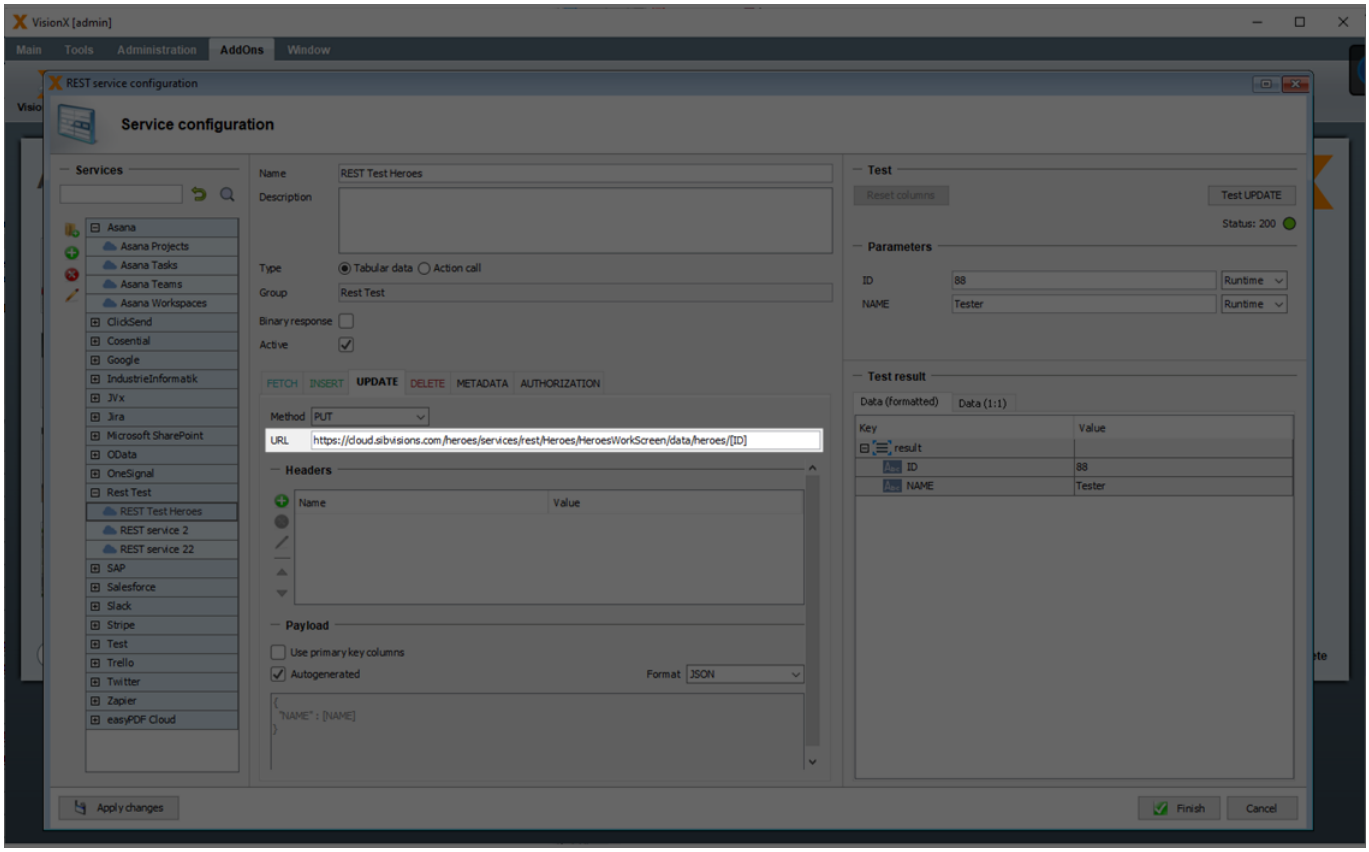



We also have to enter the URL again. Let's use the demo resource one more time:

https://cloud.sibvisions.com/heroes/services/rest/Heroes/HeroesWorkScreen/data/heroes

To define which record should be updated, we have to add the column name of the unique identifier (in this case “ID”) to the end of the URL in square brackets:

https://cloud.sibvisions.com/heroes/services/rest/Heroes/HeroesWorkScreen/data/heroes/[ID]



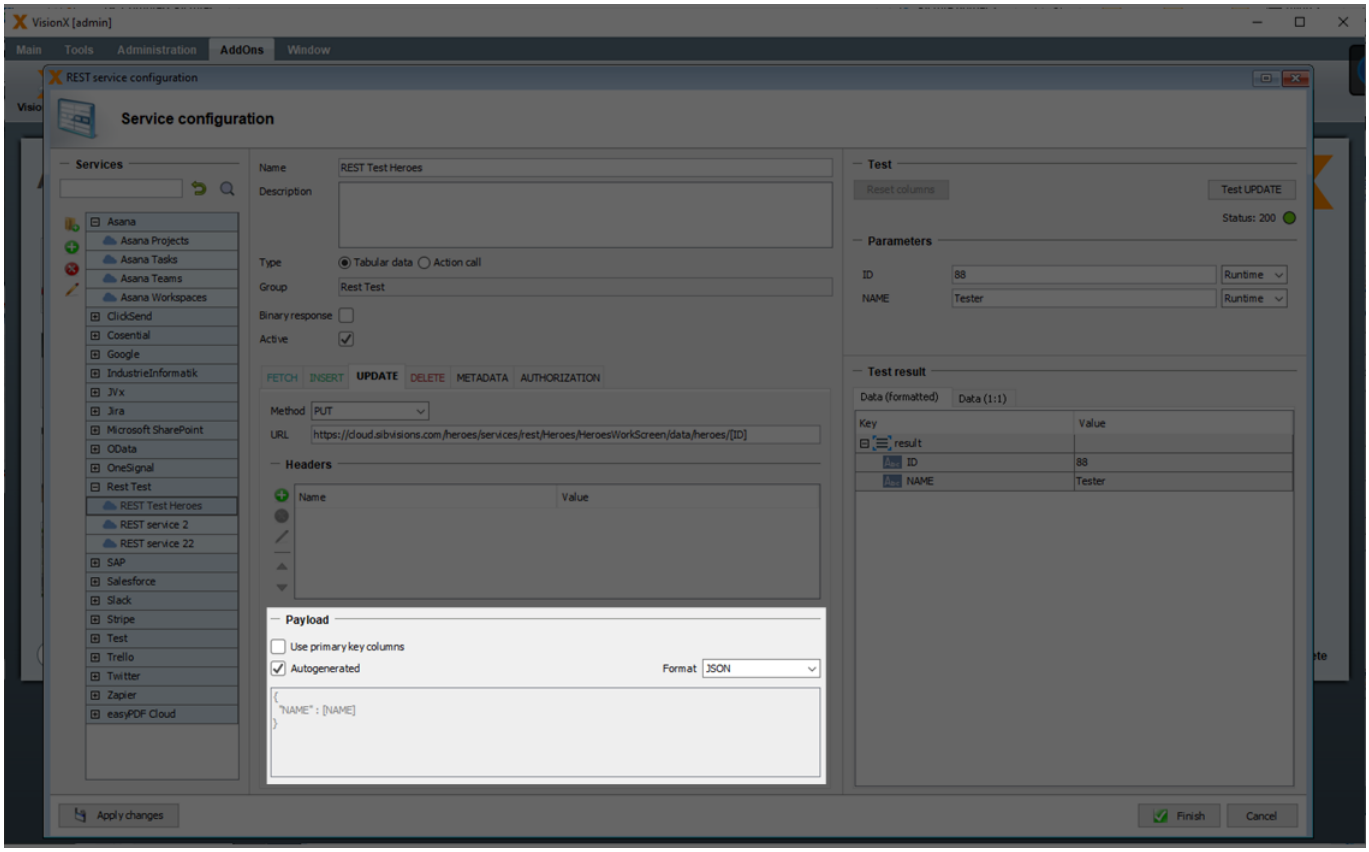

Next, we have to define the parameters for the data that we want to insert. This is done in the “Payload” section at the bottom of the screen.

The payload is autogenerated by default, using all of the columns in the table. *Note: Primary key column are only included in the autogenerated payload if we check the “Use primary key columns” box.*

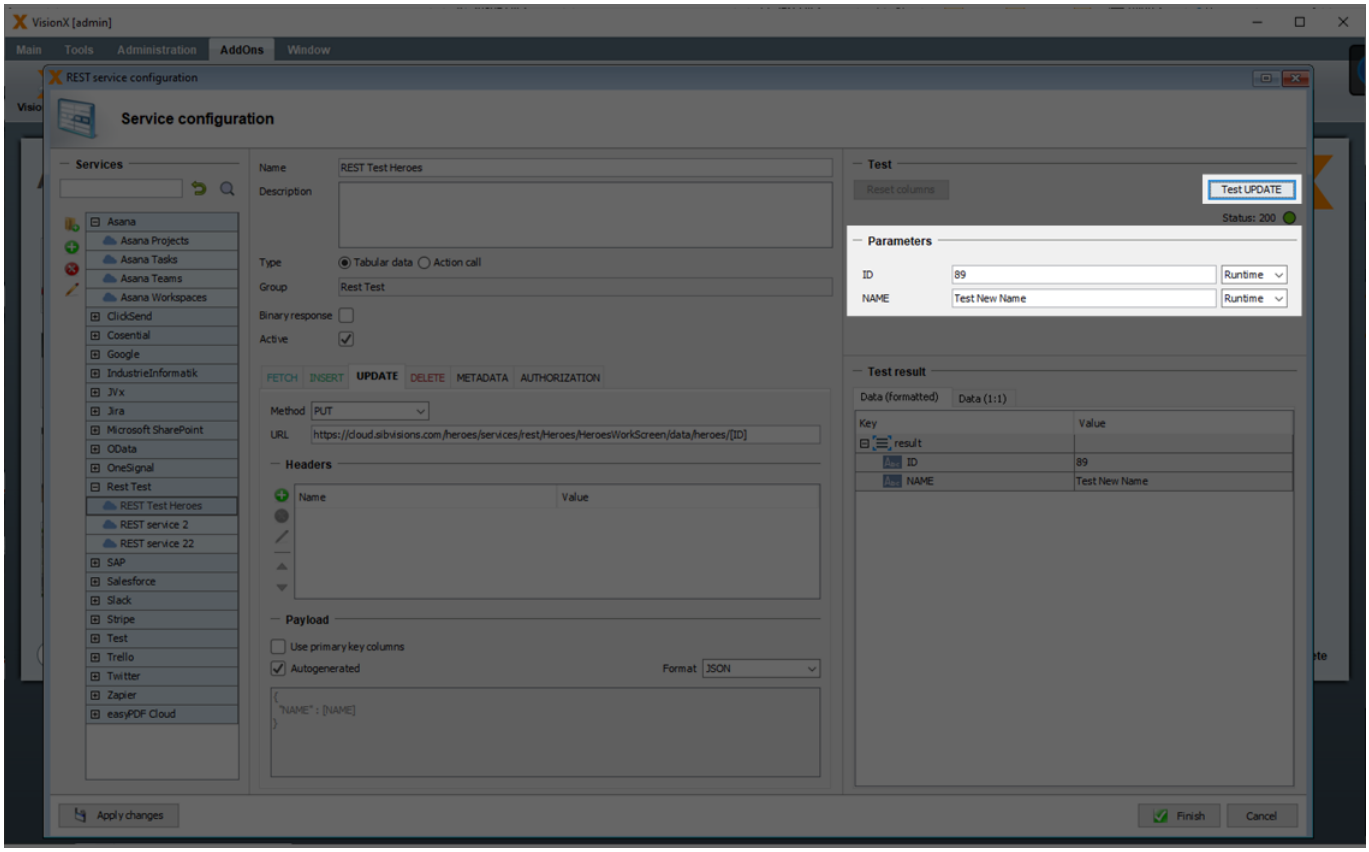
In our example, we are only going to update one column: NAME.

Only the columns that need to be updated should be included in the payload. If we want to change the payload definition, we can simply uncheck the “Autogenerated” checkbox and edit as necessary. For this example, we are not going to make any changes to the payload, so we will just keep the autogenerated text.

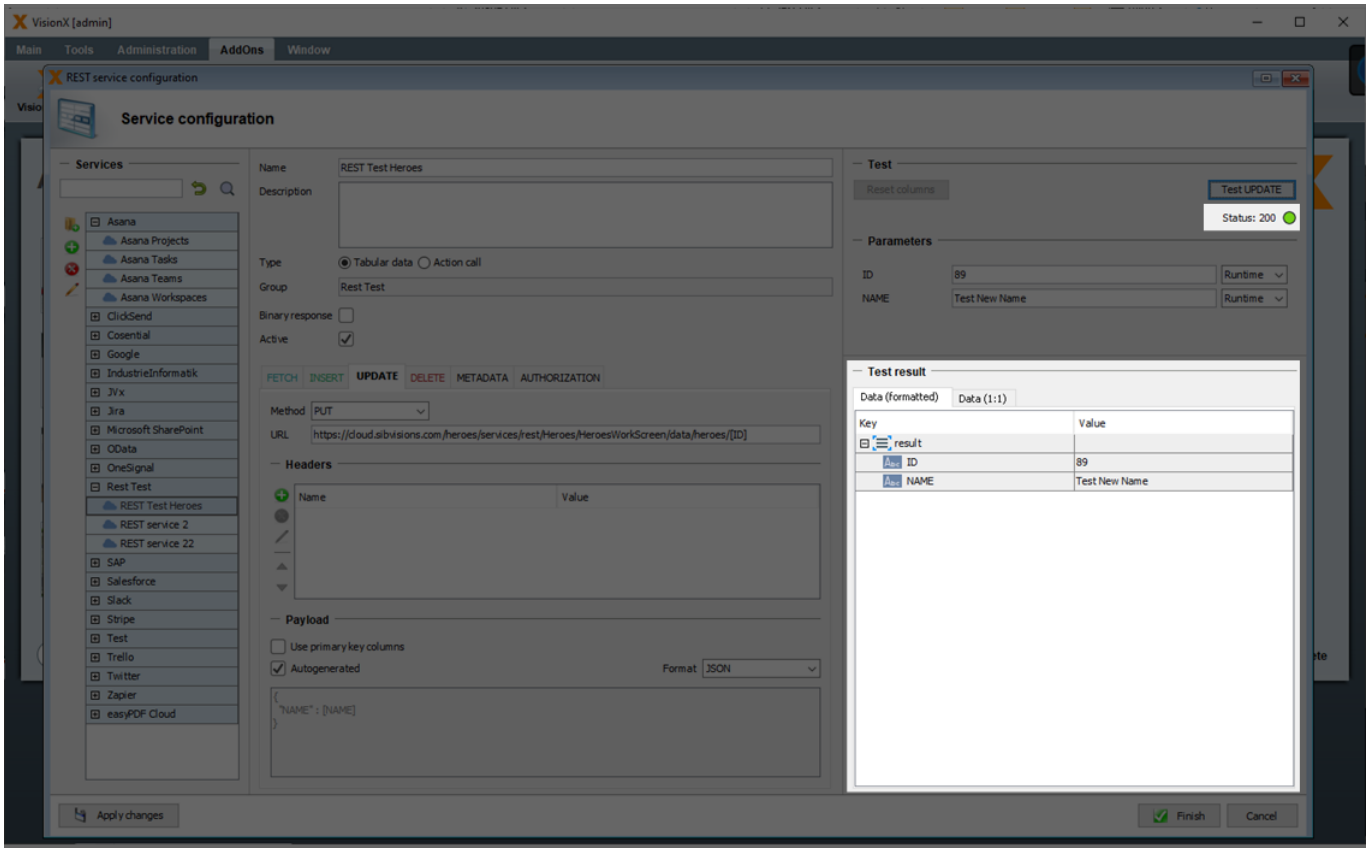
The payload format can be selected from the dropdown menu on the right. The following formats are available: Text, Javascript, JSON, HTML, XML, Binary, and Multipart.




We can now test updating a record by entering an ID and Name under “Parameters” on the right side of the screen, and then pressing the “Test UPDATE” button.




The 200 status code means that the record was successfully updated. We can see the result with the updated name in the “Test result” window.

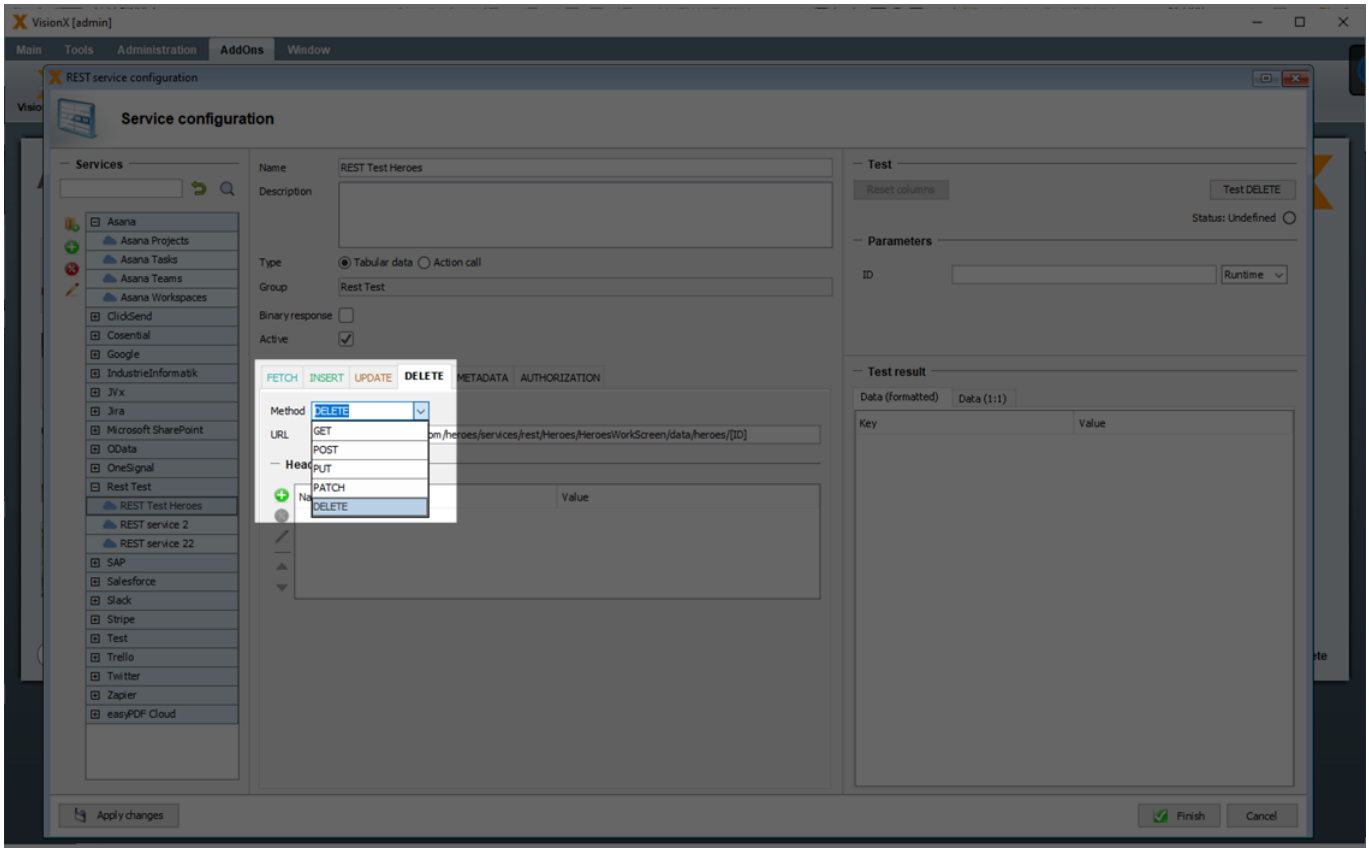



*Note: REST services vary in their requirements for specifying columns when sending an “UPDATE” request. Please refer to each service's REST documentation for details.*

# DELETE

The “DELETE” request is used to delete a database record; it is usually done using the HTML Delete method.

To delete a record, click on the “Delete” tab and select the “DELETE” method from the dropdown menu.

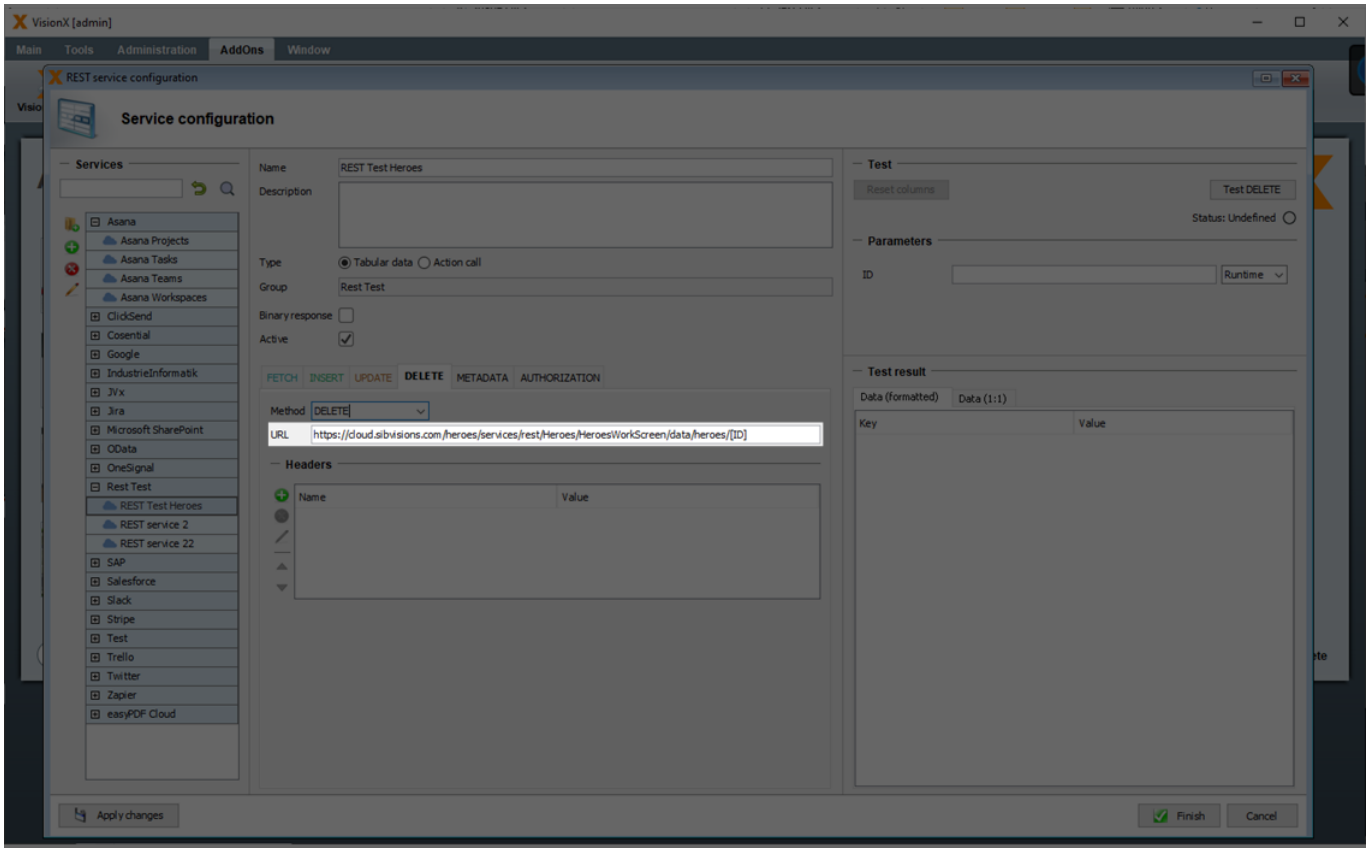



We now have to enter the URL. We are going to use the demo resource again:

https://cloud.sibvisions.com/heroes/services/rest/Heroes/HeroesWorkScreen/data/heroes

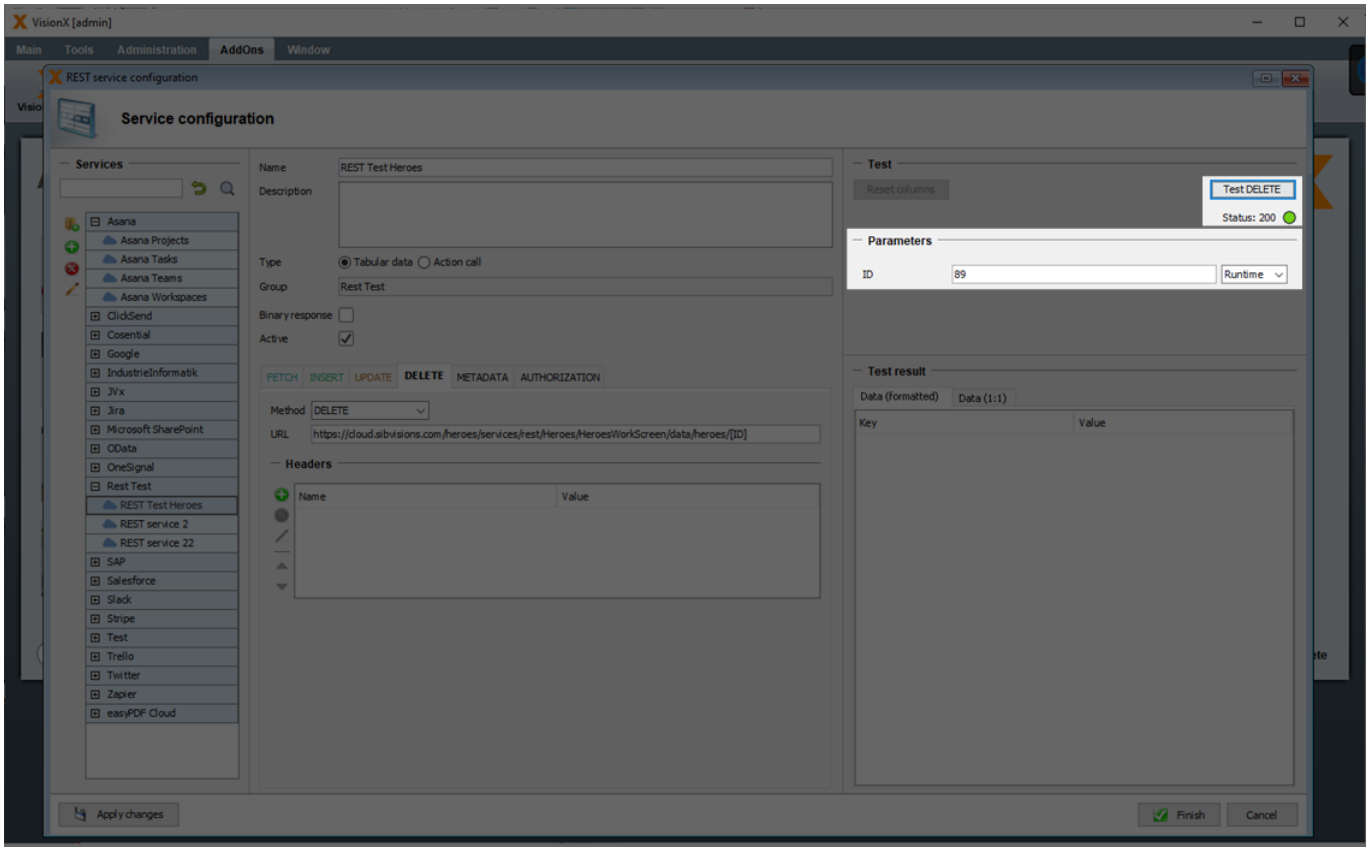
To define which record should be deleted, we have to add the column name of the unique identifier (in this case “ID”) to the end of the URL in square brackets:

https://cloud.sibvisions.com/heroes/services/rest/Heroes/HeroesWorkScreen/data/heroes/[ID]




Next, we simply have to identify the record we want to delete by entering its ID under “Parameters”. Press “Test DELETE” to test the request.

The 200 status code means that the record was successfully deleted.



# METADATA

In addition, metadata can be defined on the “METADATA” tab.

# AUTHORIZATION

On the “AUTHORIZATION” tab we can select the authorization method and enter our credentials.

*Note: Please note that the authentication methods are different for each vendor, so read the instructions of the REST service provider carefully!*

The following authorization methods are available:

- **Basic**
  Basic Authentication is the classic method to authenticate with user and password.
- **OAuth1**
  OAuth 1 is a more secure authentication metghod than the classic basic authentication. Due to the further development to OAuth2 it is only rarely used.

| Setting | Description |
|---|---|
| **Authorization endpoint** | Its the url to start the authentication. |
| **Client ID** | The client id of the registered app that is allowed to access. |
| **Client secret** | The client secret of the registered app that is allowed to access. |
| **Access token** | The acces token of the registered app to start the auhtentication. |
| **Access token secret** | The acces secret of the registered app to start the auhtentication. |

More info see https://de.wikipedia.org/wiki/OAuth#

- **OAuth2**
  OAuth 2 is a more secure authentication method than the classic basic authentication.

| Setting | Description |
|---|---|
| **Authorization endpoint** | Its the url to start the authentication. |
| **Access token endpoint** | Its the url to get the access token after the authentication. |
| **Refreh token endpoint** | Its the url to refesh the access token, after its expired. |
| **User** | The user of the registered app that is allowed to access. |
| **Password** | The password of the registered app that is allowed to access. |
| **Client ID** | The client id of the registered app to start the auhtentication. |
| **Client secret** | The client secret of the registered app to start the auhtentication. |
| **Scope** | The scope specifies which functions may be called. You could compare it with the roles in a user administration. |

https://de.wikipedia.org/wiki/OAuth

- **Shared Key**
  Shared Key is a very common mehode. Usually you only need to copy the **key name** and **key**

**value** from your Saas account.

- **Bearer Token**
  Bearer Token is a very common method. There are different interpretations for each vendor to create the initial Bearer Token. We support - **Google, Microsoft, Dropbox, Salesforce und Adobe Sign**.

  If you check “Generate”, then you can choose the vendor. Then fill in the client ID of the registered app, the redirect URL, e.g.: the url of your installed app or localhost, and the scope, i.e. which functions you want to call. After that click on “Get authorization code” to generate the authorization code. This will be automatically filled in the “Authorization code” field. Please enter the Client Secret and click on “Generate Token” to generate the Bearer Token.

  ***Note - If you use another vendor, you have to create the Bearer Token via the vendor interfaces and enter it here as Bearer Token.***

# Using Data in an Application

Once the connection has been configured, you can use the data in VisionX just like a database table. Let’s take a look at how that is done.

First, we open the “Contacts” application that is pre-installed in VisionX.

We will then add a new screen and name it “REST Test”. Click “Next”.

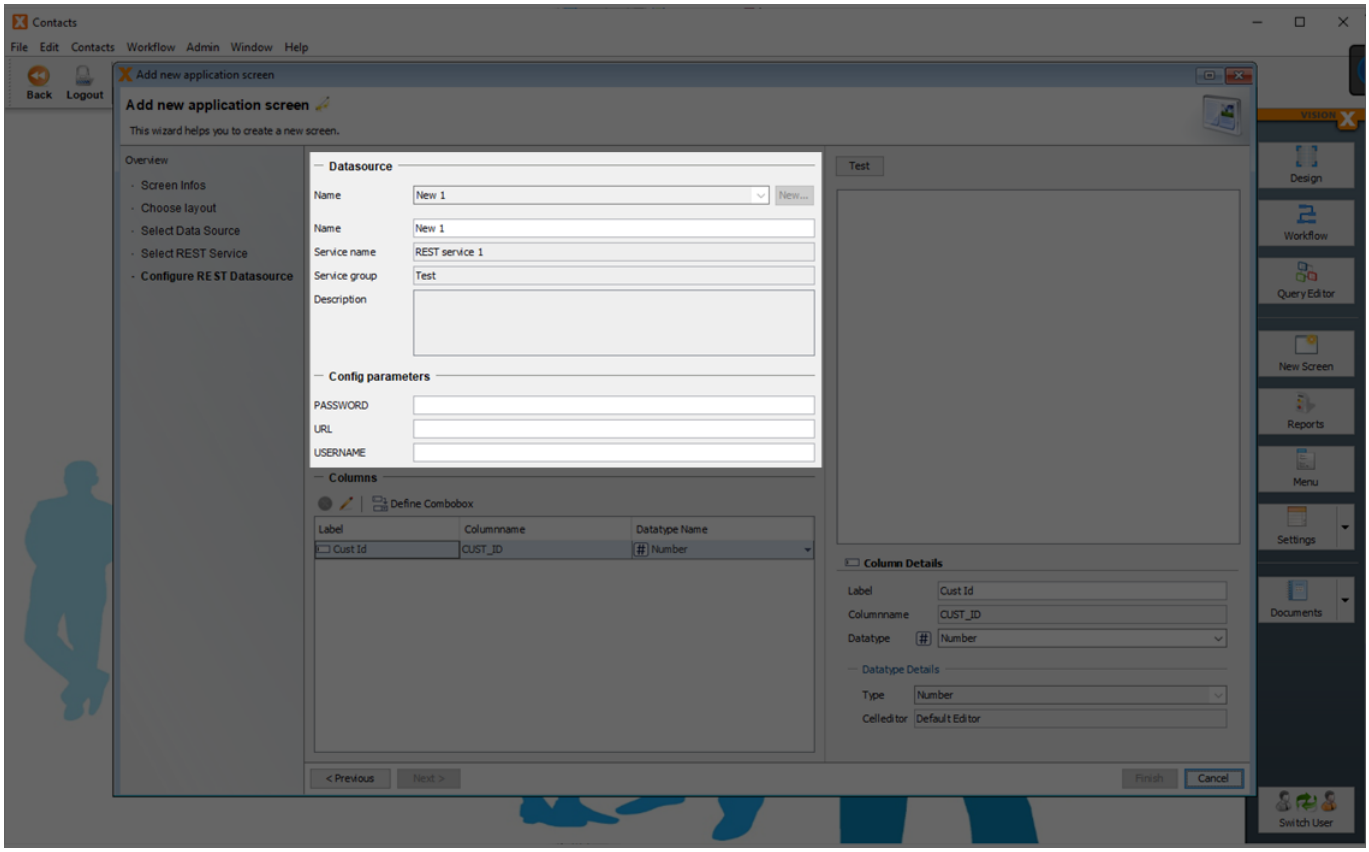
For the layout, we select “Table with detail form” and “Universal layout” and click “Next”.

For the data source, we select “REST Service” and click “Next”.

On the next screen, we select the REST service we just configured (Rest service 1) and click “Next”.

On the following screen, we can enter a name for the data source (optional).

Also, if any config parameters were defined during configuration, they can be entered here. Common config parameters are username, password and the resource URL (or parts of it), which are all shown in this example.




Click on the “Test” button at the top right to test the connection. The results will show up in the table on the right side.

If necessary, column details can be edited in the “Columns” table and the “Column Details” section on the bottom right.

Click “Finish” to create the new application screen.

You can now use the REST data just like any other data table in VisionX.

# Action Calls

We can also use REST services in VisionX actions. To do that, we first define a new REST service and chose “Action call” as the type.

We then add the authorization details, same as before (admin/admin).

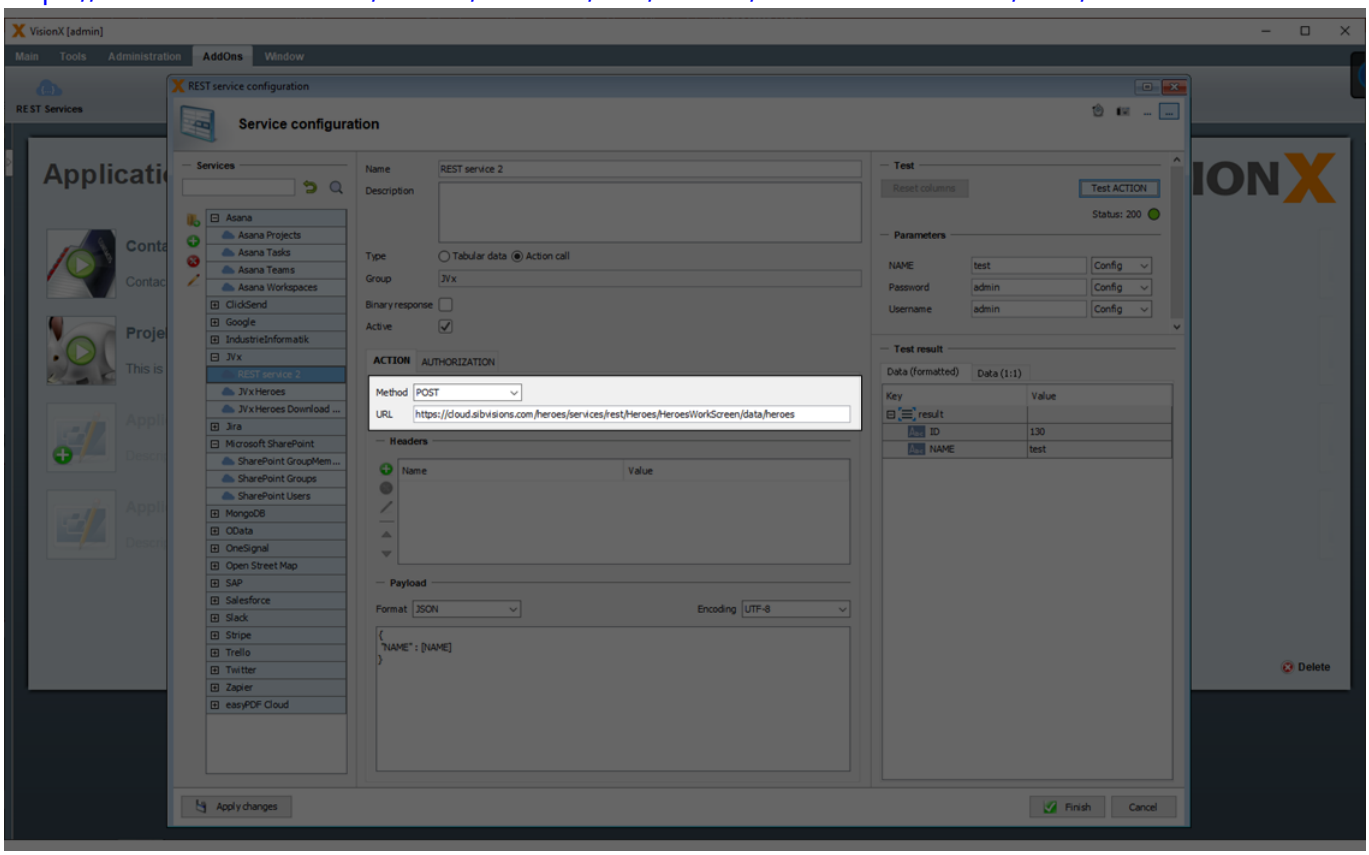
We then select the method and add the URL, just like in the previous examples.

*Note: The GET method should never be selected for action calls. Actions are typical use a POST, PUT or DELETE method to execute a task or insert, update or delete records.*

In our example we use the POST method to create a new record

We now have to enter the URL. We are going to use the demo resource again:

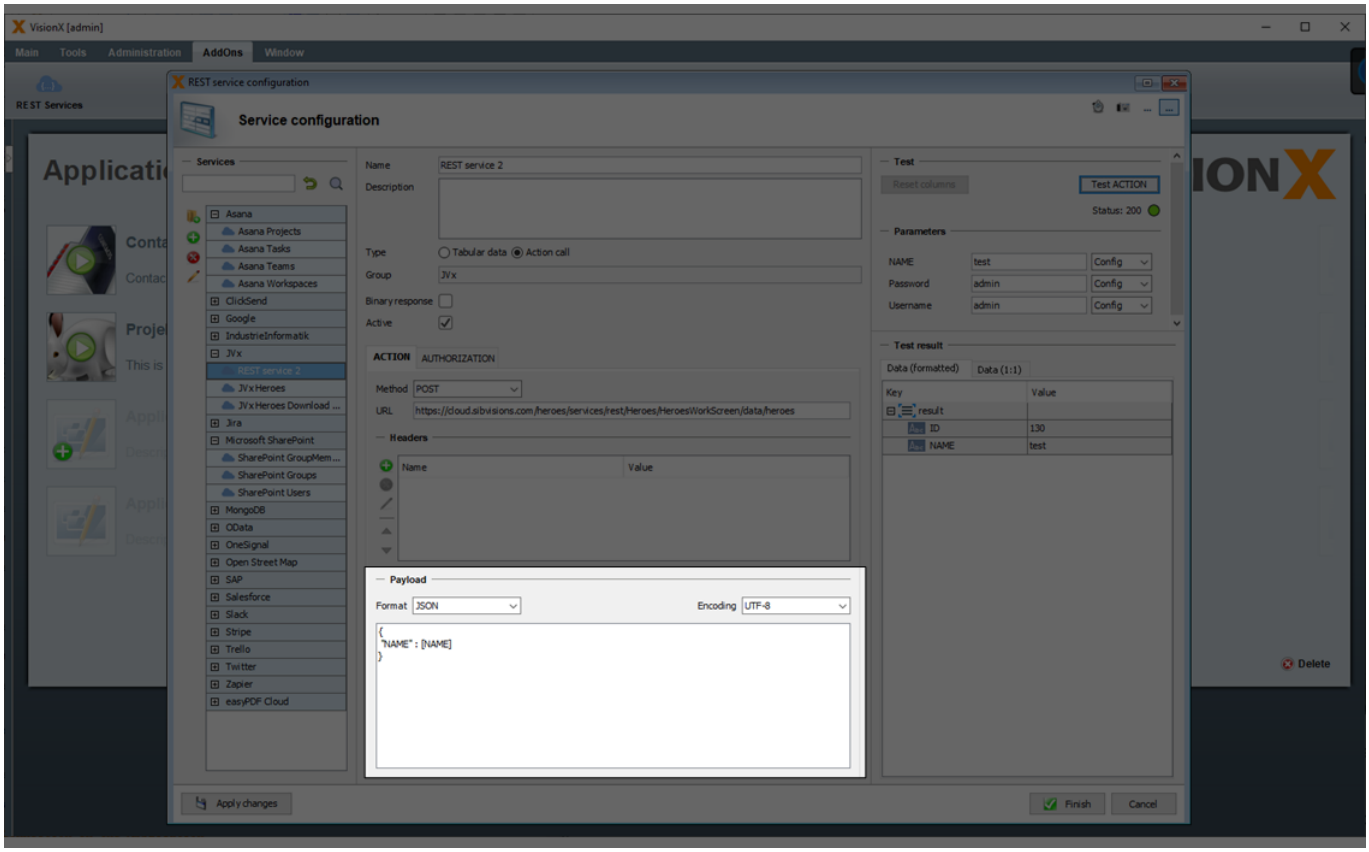
https://cloud.sibvisions.com/heroes/services/rest/Heroes/HeroesWorkScreen/data/heroes



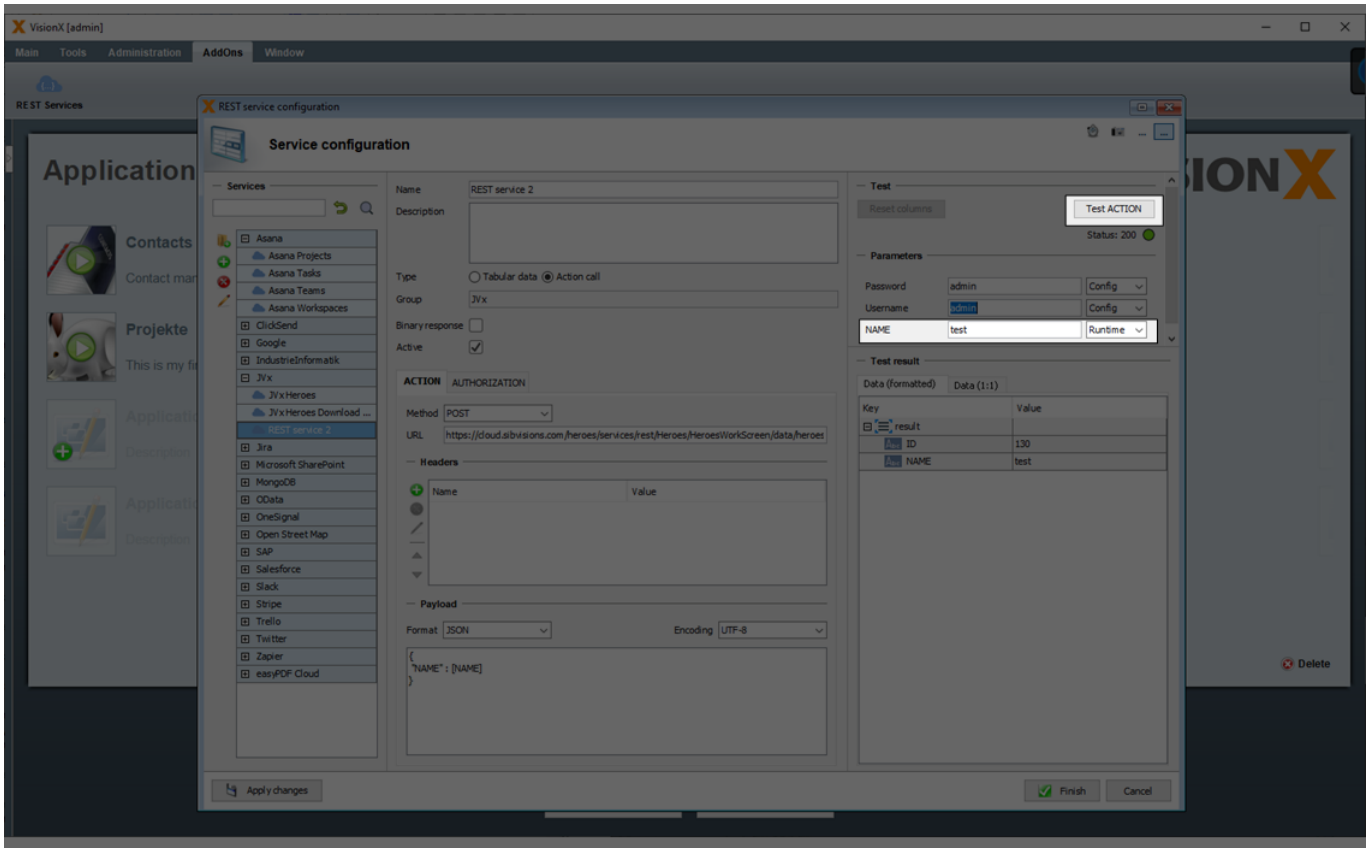

Next, we have to define the parameters for the data that we want to insert. This is done in the “Payload” section at the bottom of the screen.

In our example, we are only going to include one column: NAME. We use this payload ~~json { “NAME” : [NAME] }~~

The payload format can be selected from the dropdown menu on the right. The following formats are available: Text, Javascript, JSON, HTML, XML, Binary, and Multipart.




***Note: Please define the parameter “NAME” as runtime parameter. This will then be used as a parameter for the VisionX action. See next chapter.***




We can now test inserting a record by entering a Name under “Parameters” on the right side of the screen, and then pressing the “Test Action” button, and click “Finish”.

Now let’s go back to our application to test the action call. We first add a button to our application screen and name it “REST Call”.

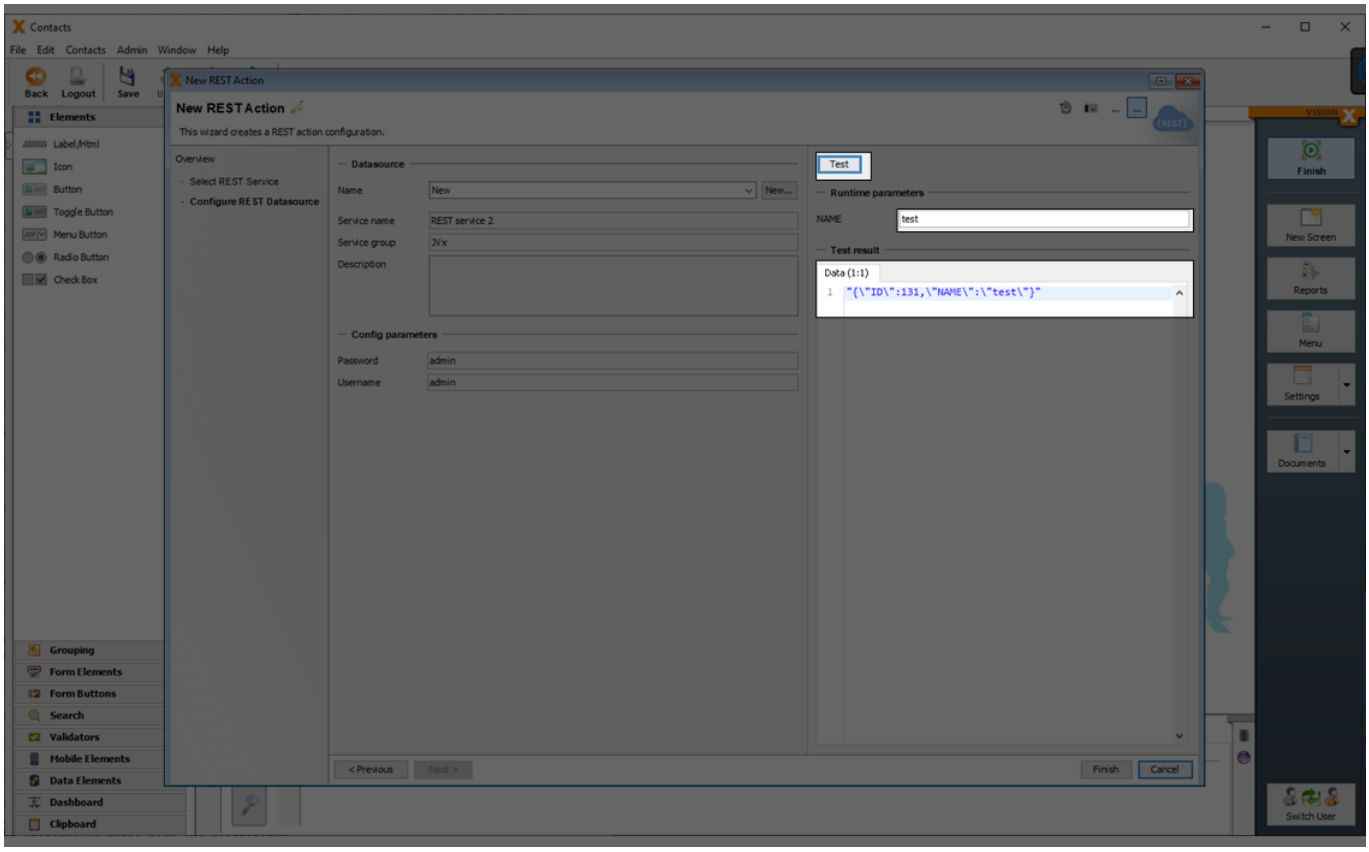
We then define the action for the button by clicking on the “Pencil” icon and selecting “Create Action” at the bottom of the popup menu.

On the “Action” tab, we select “Call REST Action” from the dropdown menu.

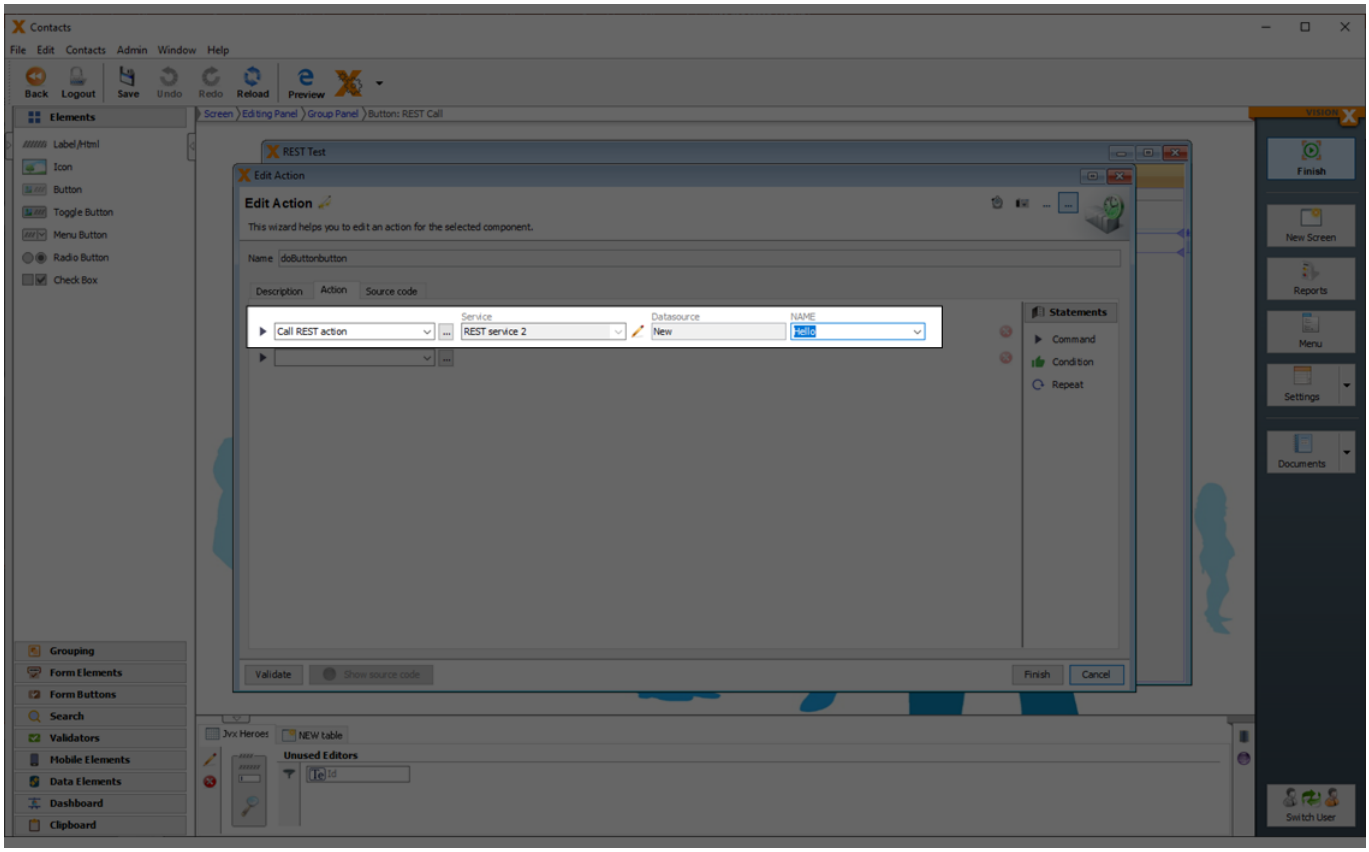
We then select the REST service we just defined using the pencil icon under “Service”.

The next screen shows the list of REST services that have been defined with the type “Action call”. In our example, there is only one (“REST service 2”), so we select it and click “Next”.

Please fill the runtime parameter “NAME” and click on “Test” to test the REST action to add a new record. Then click Finsh.



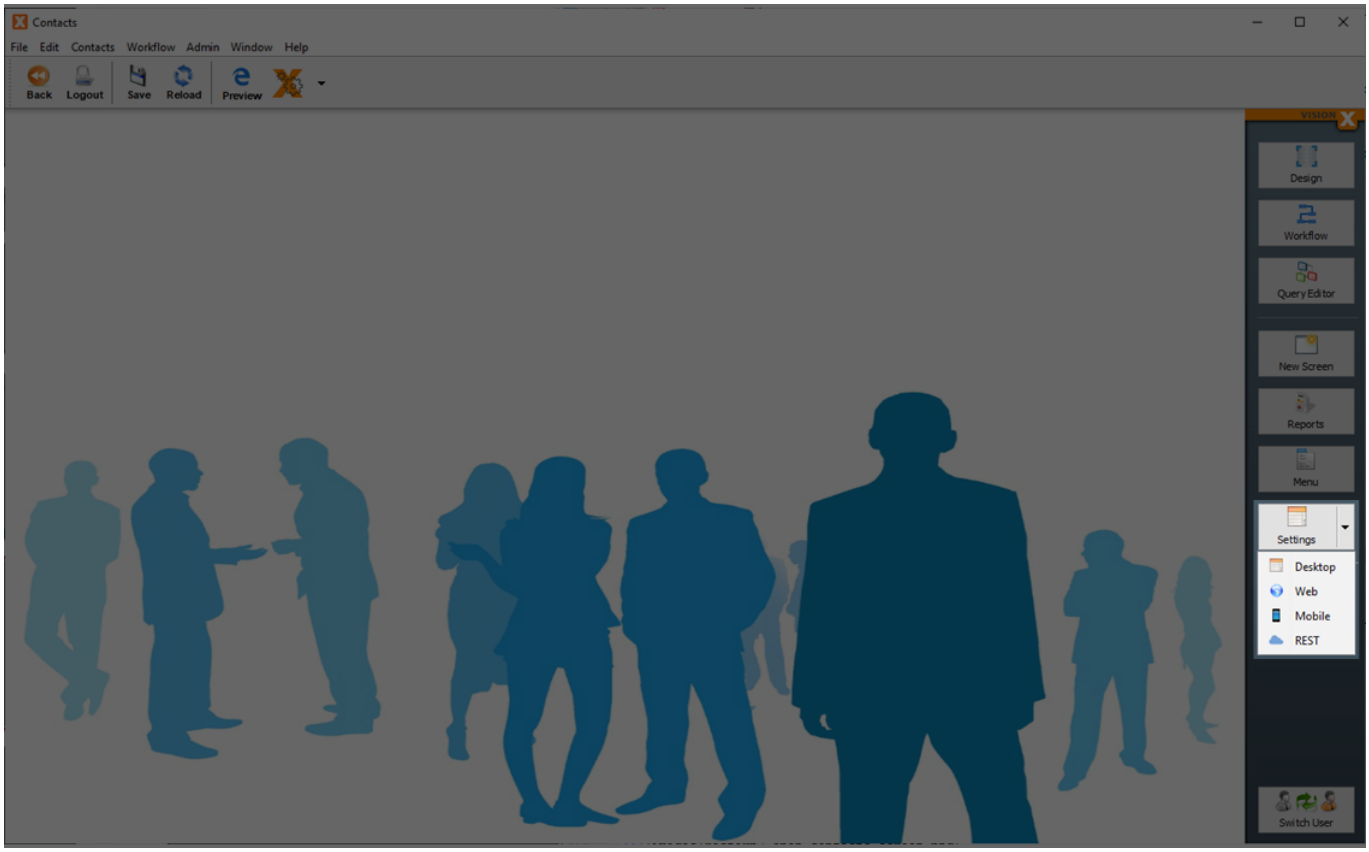

Now fill the “NAME” parameter of the REST action with the value “Hello”.



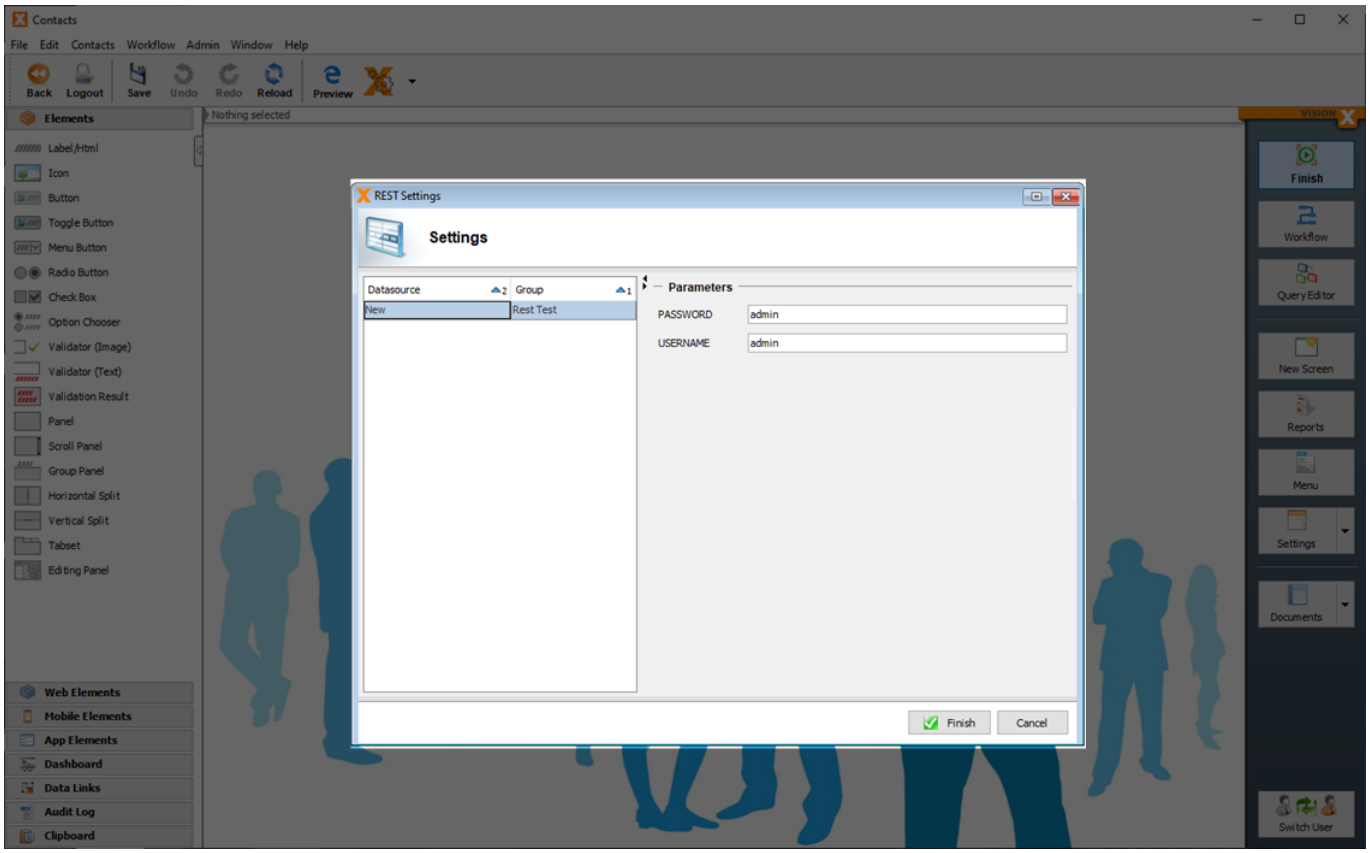

Then click Finsh and test the app. A new record with the “NAME” “Hello” will be created on button click. To see the new record in the table click on Reload in the application.

## Connection in the Application

The REST connection can be configured in the application by opening the “REST” item on the “Settings” menu.




Here we can select the datasource and change the config parameters.



# Exposing Data via REST

VisionX application data can easily be exposed using our REST documentation. In this example, we are going to use the “Contacts” application that comes preinstalled with VisionX. We first open the application:

When we open the “Contacts” application screen, we see a list of contacts with addresses and other details. This is the data we want to expose.

To create the REST documentation, we first click on the dropdown arrow next to the “Documents” button on the menu on the right side and select “REST Services”.

Select “Create Swagger documentation” and click “Next”.

At the next step, we can select the application screens. All of them are checked by default, so we can just click “Finish” to create the documentation for all screens.

A browser window will open with the documentation

# GET

Let’s start with a GET REST call. Scroll down to “Contacts” and click on the “GET” line to expand it.

Then click on the “Try it out” button on the right side of the screen.

We could now enter parameters, but we will leave those blank for now and scroll down and click the “Execute” button.

We can now see the contents of the “Contacts” screen in the response body.

We also get the same response if we take the “Response URL” and paste it in the browser address line.

# PUT

We can also update a record using a PUT call. We first expand the PUT line in the “Contacts” section and click the “Try it out” button.

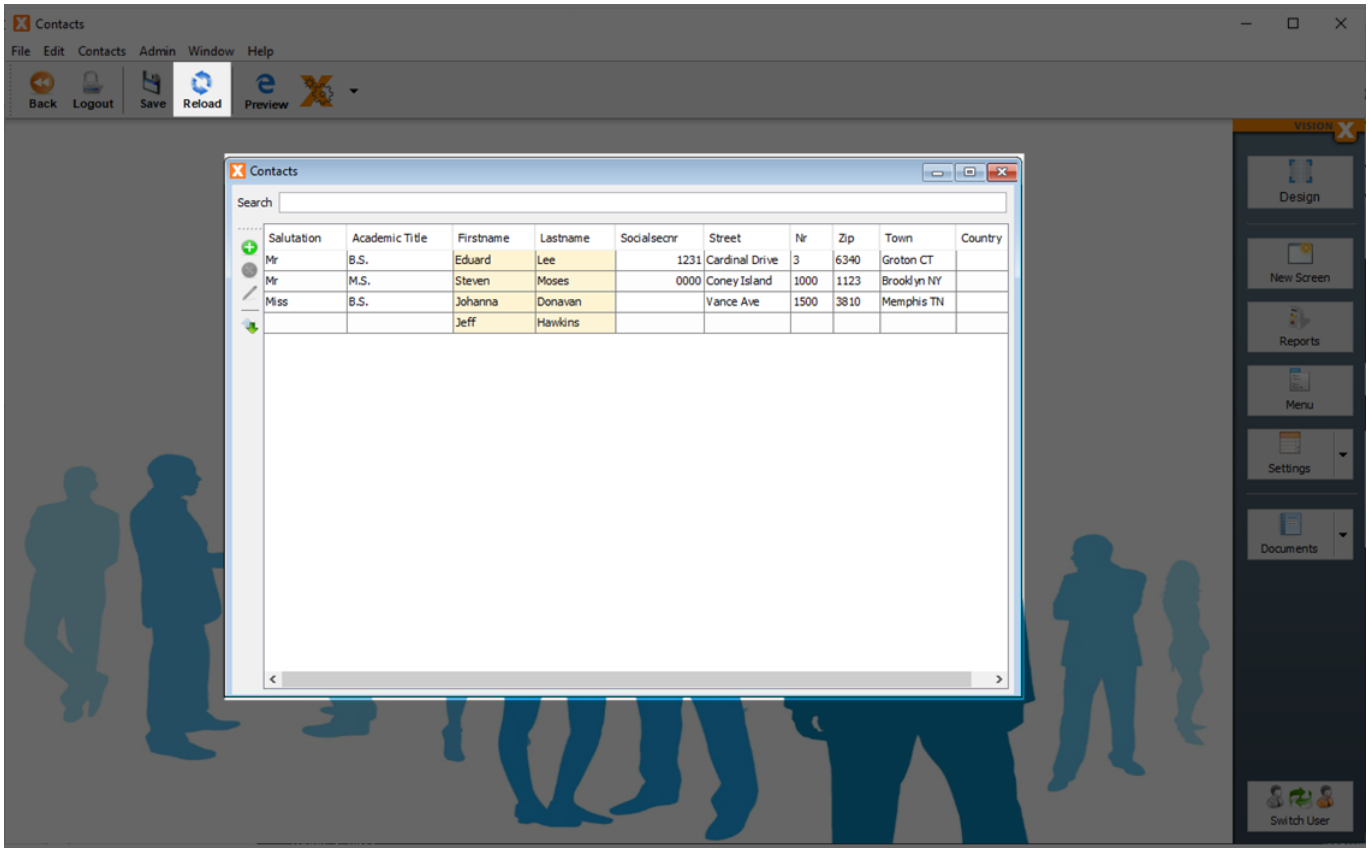
We first enter the primary key of the record we want to update. In this case, the primary key is the ID column, and we’ll pick “2” for this example.

In the “Body” section we can see all the fields on the “Contacts” screen.

To keep this example simple, we are only going to update the “First Name” field, so we can delete the rest of them. We also enter a new first name for that record.

After clicking “Execute”, we can see that the first name has been updated.

We can also see the updated record in VisionX after hitting the “Reload” button.




# DELETE

We can delete a record using a DELETE call. To do that, we click on the “Delete” line in the “Contacts” section and hit the “Try it out” button.

Then we simply enter the ID of the record we want to delete and hit “Execute”.

The response will simply show the 200 code, which stands for “Successful operation”. We can again hit “Reload” in VisionX to confirm that the record has been deleted.

# POST

The POST call is used to insert a new record. We first click on the “POST” line in the “Contacts” and then the “Try it out” button.

We can again see the list of all fields on the “Contacts” screen.

We do not have to enter data for each field, but we have to make sure that data for the required fields is entered. Required fields are identified in VisionX by yellow highlights in the table.

We can also see the required fields in the data model in Design Mode (the “Mandatory” column).

Keeping that in mind, we now enter the data for the required fields and click “Execute”.

In the response body we can see that the new record has been created with the data we entered.

And again, we can hit “Reload” in VisionX to confirm that the new record is there.