The Lianja 11 AI Assistant

Executive Summary

Lianja 11 is a comprehensive **AI Powered**, *data-centric* Application Platform as a Service (APaaS) designed to significantly boost developer productivity and streamline the entire lifecycle of business application development. The platform emphasizes visual design, rapid development, and flexible deployment across a multitude of platforms including Windows, Mac,

and Linux, as well as various client devices such as Desktop, Web, Mobile, Progressive Web Apps (PWA), and Electron Apps. A cornerstone of Lianja's value proposition is its unparalleled multi-language support, embedding JavaScript, LianjaScript (fully compatible with Visual FoxPro), Python and TypeScript. This unique integration allows developers to leverage their existing skillsets without the need to learn a new language, thereby minimizing retraining efforts and accelerating project timelines. This capability stands as a critical differentiator in a market where many platforms often restrict developers to a single language or a limited set of tools.

Lianja adopts a unique "Integrated **No-Code**, **Low-Code** and **Pro-Code** Tools" philosophy. This means it caters to a broad spectrum of users: from business domain experts (citizen developers) who can visually design and build applications with minimal or no coding using the intuitive **Lianja App Builder**'s drag-and-drop interface, to professional developers who require the flexibility to extend functionality through code when needed. This hybrid approach allows for rapid prototyping and iterative refinement while ensuring deep customization capabilities.

Other notable features include a comprehensive set of UI classes that enable the creation of modern-looking applications, all of which can be easily customized and skinned using CSS, eliminating the need for hardcoding visual styles. A built-in, cross-platform web browser component (based on Webkit) and integrated server page processing engines for LianjaScript (.rsp), JavaScript (.jssp) and Python (.pysp) allow server-side logic within Lianja Apps without external installations. The **Lianja Cloud Server** embeds LianjaScript, the Google-developed V8 JavaScript engine and Python, providing robust support for server-side scripting. Advanced security features, including user roles and permissions (assignable at App, Page, Section, and Field levels, with static or dynamic application), Row Level Security (RLS), and Dynamic Data Masks (DDM) control data visibility and access. A built-in Data Dictionary for managing business rules (validation, permissions, calculated fields), Chronological Data Versioning (Timelines), and audit trails for regulatory compliance are also present. **Lianja SQL Server** is highly compatible with *Microsoft T-SQL* and is embedded into both the Lianja App Builder and the Lianja Cloud Server. Finally, the Lianja App Center serves as a secure, centralized hub for deploying and provisioning applications to end-users across desktop, web, and mobile devices, with customizable corporate branding.

Build AI powered Apps for the future

In today’s data-driven world, the ability to efficiently gather and process information is paramount for the success of artificial intelligence (AI) applications.

The next step in AI will be applications.

Lianja AI Assistant has full support for building custom AI agents that integrate seamlessly into your Lianja Apps.

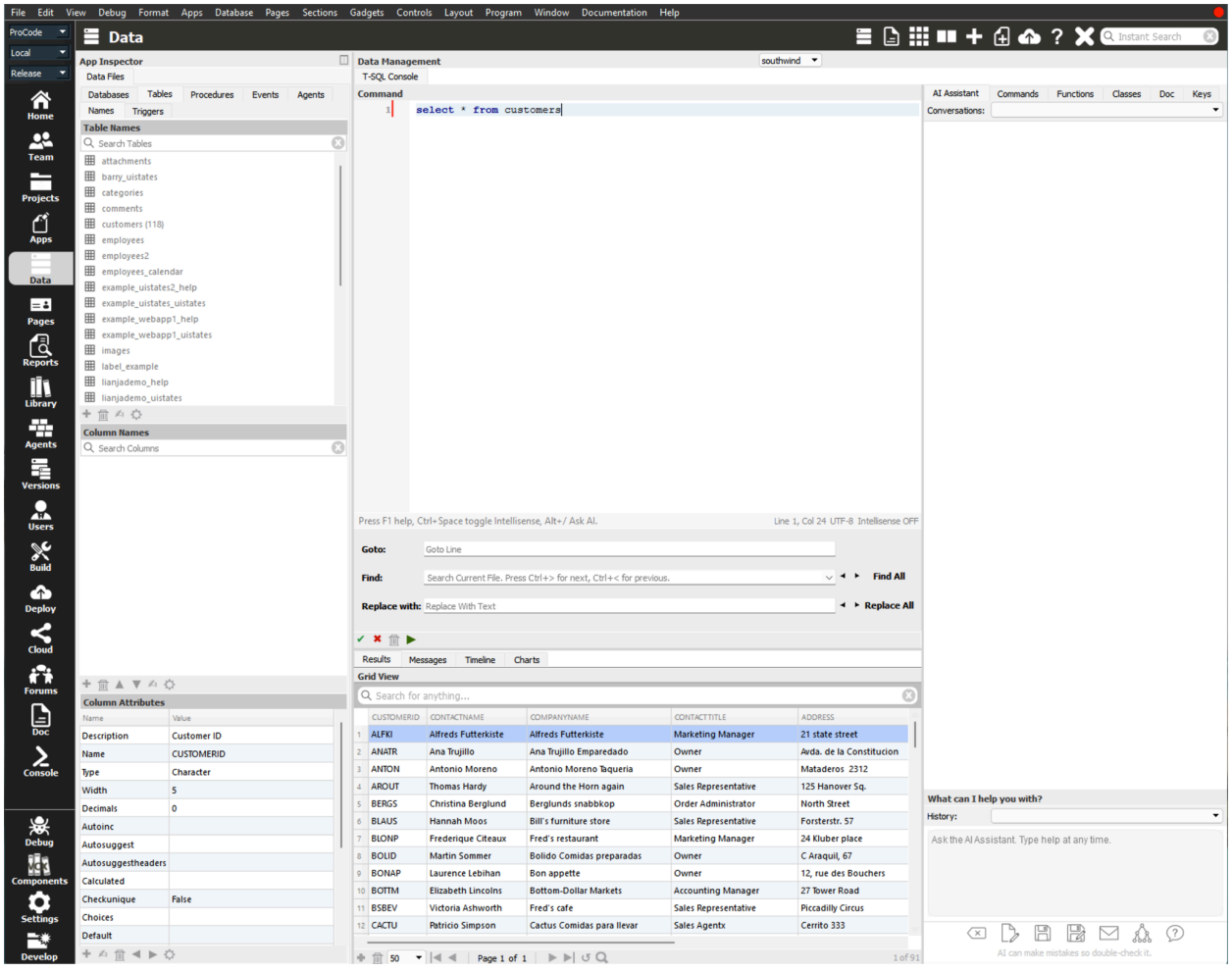
AI powered database management

The Lianja AI Assistant provides AI powered natural language queries in the Data Workspace of the App Builder and any page within an App. Desktop, Web and Mobile are all supported. Additionally and uniquely, Lianja 11 has T-SQL compatibility which has been extended with AI Natural Language Query capabilities (NLQ) using the PROMPT command. This new SQL command returns the result set from a SQL SELECT statement generated by AI when you ask it something in natural language.

The Lianja AI Assistant makes extensive use of T-SQL on the client side as it interacts with AI providers such as OpenAI.

The Lianja T-SQL database engine is embedded within the Lianja client at both development and runtime providing fast and reliable response times for even the most demanding Natural Language Queries asked by a user.

As you can see from the screenshot below, the AI Assistant is displayed on the right of the window.



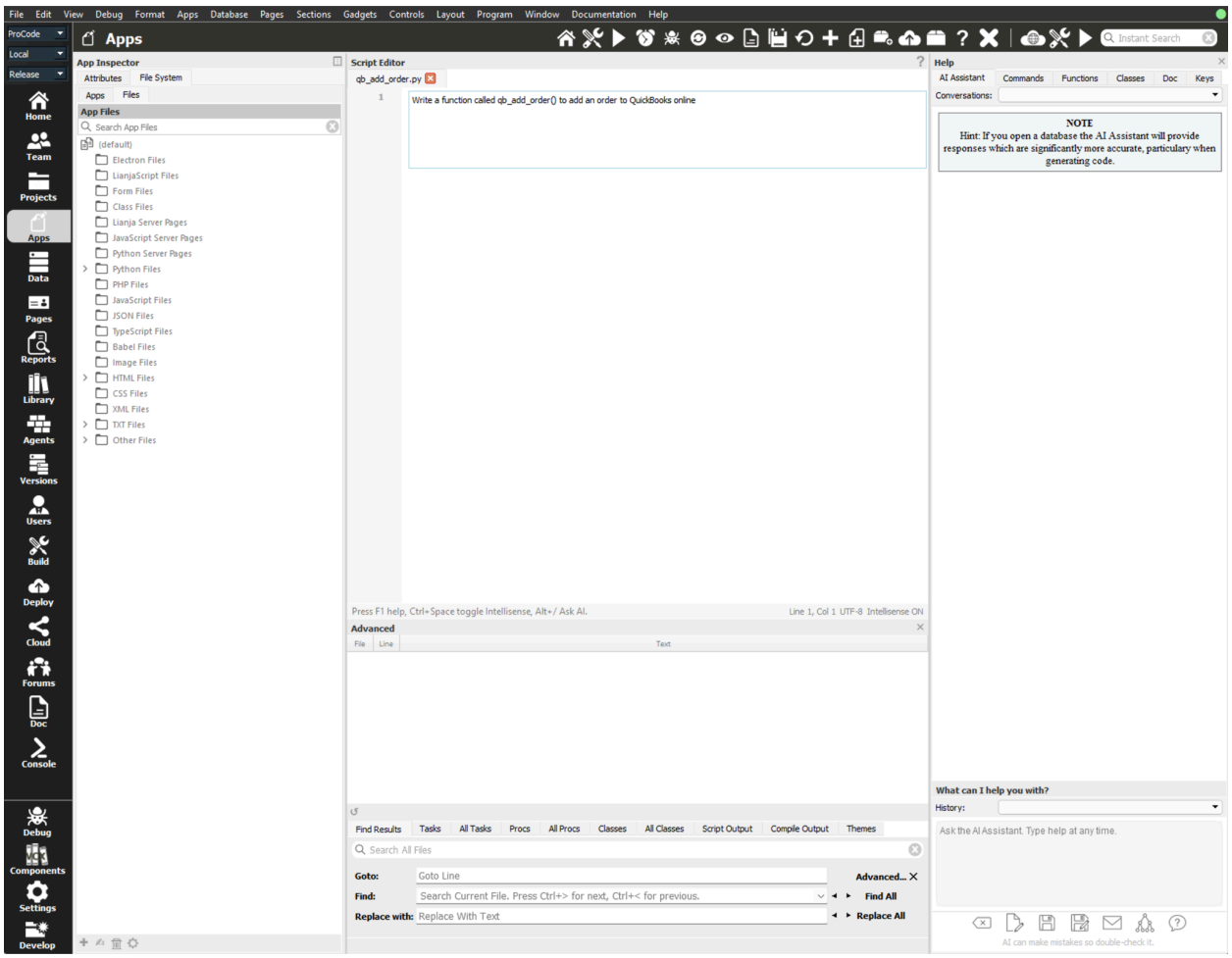
AI powered script editing

The AI Assistant is an integral part of the script editor in the Apps, Library, Agents and Data workspaces, reducing coding by an order of magnitude. Press Alt+/ and ask the AI Assistant to create the code for you and insert it into the file at the current position.

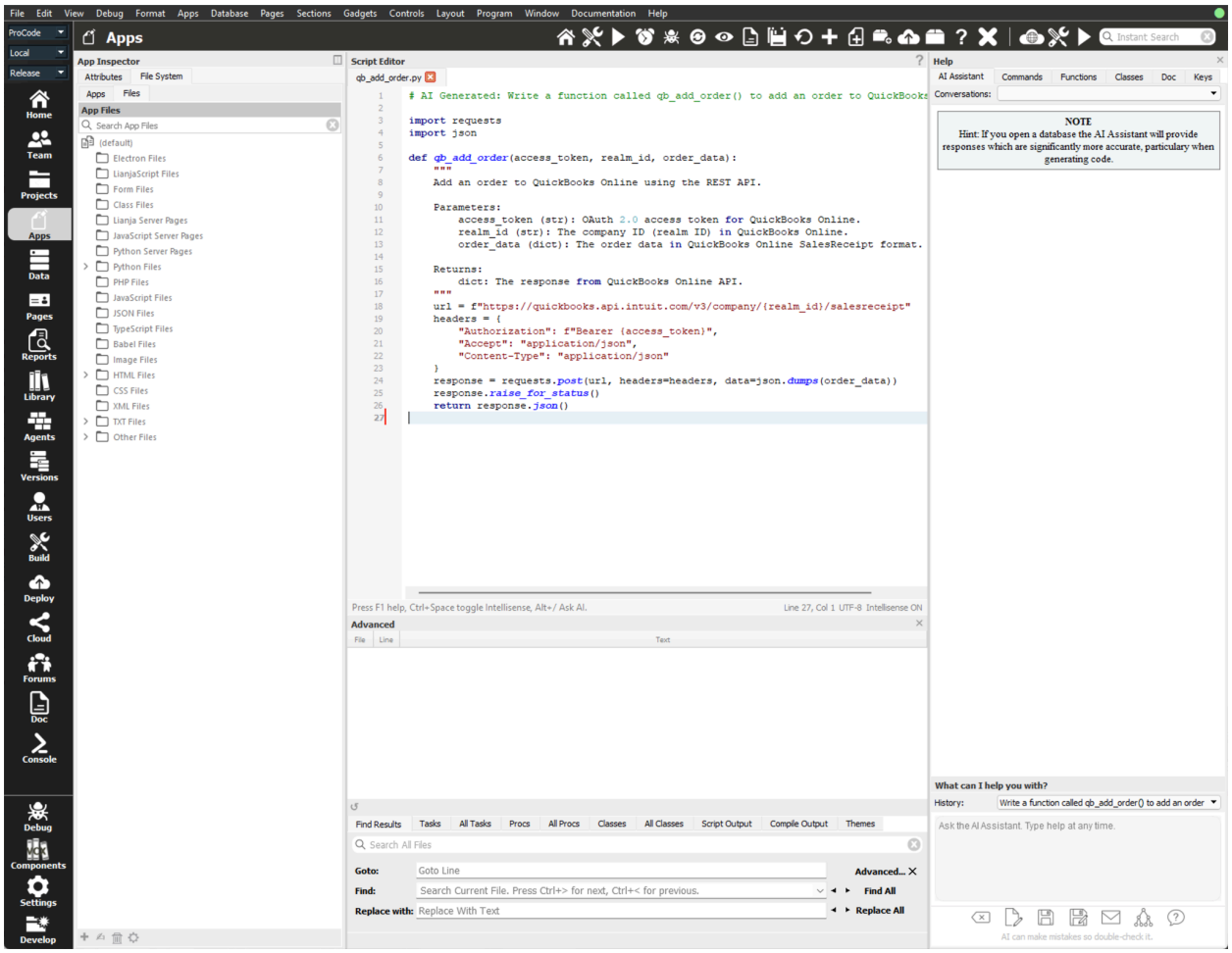
Working in the Apps Workspace

In the Apps or library workplace create a new LianjaScript or Python script called qb_add_order Press Alt+/ then type:

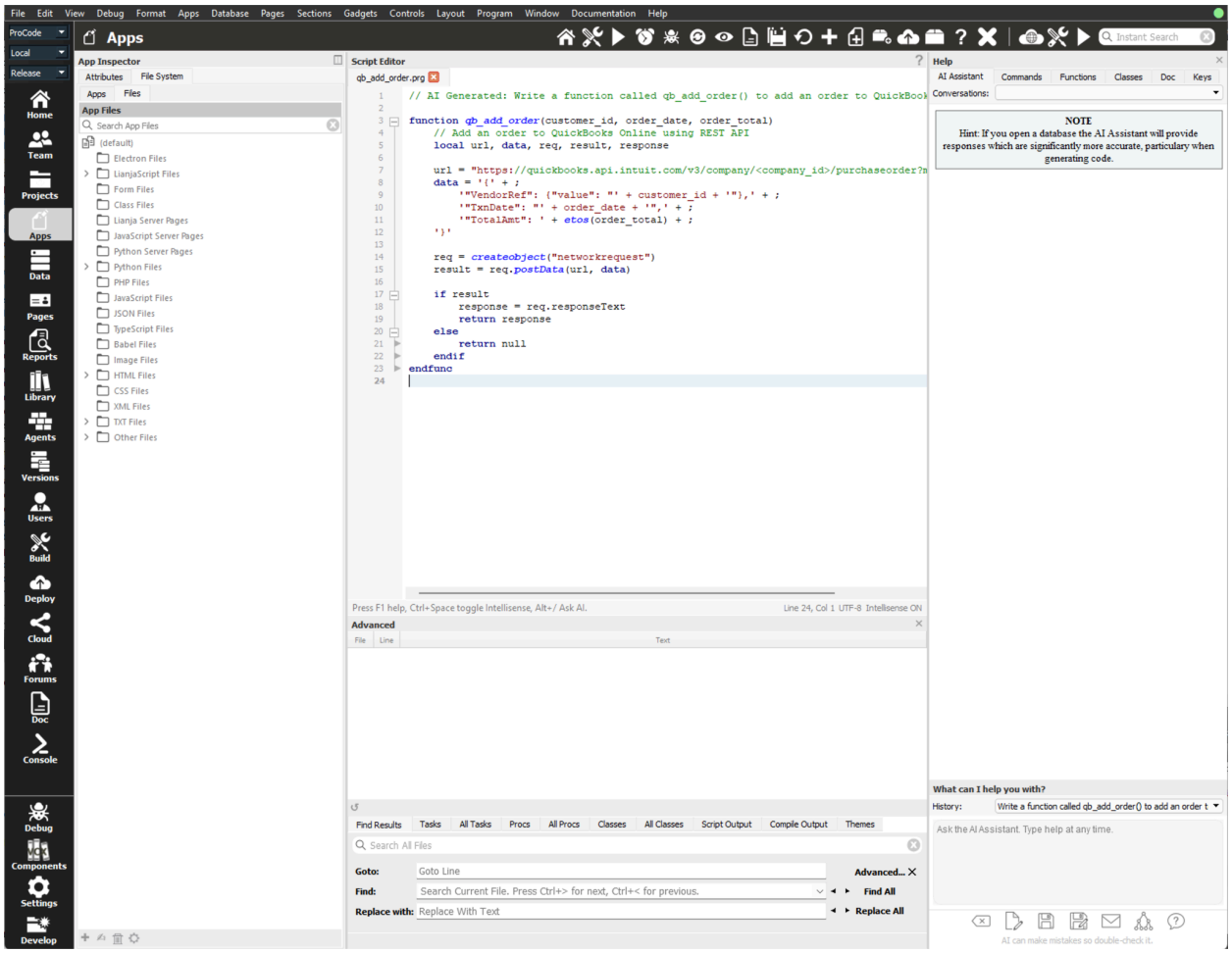
```
Write a function called qb_add_order() to add an order to QuickBooks online
```



Press Return and the Python code will be generated for you.



Similarly in a LianjaScript file.

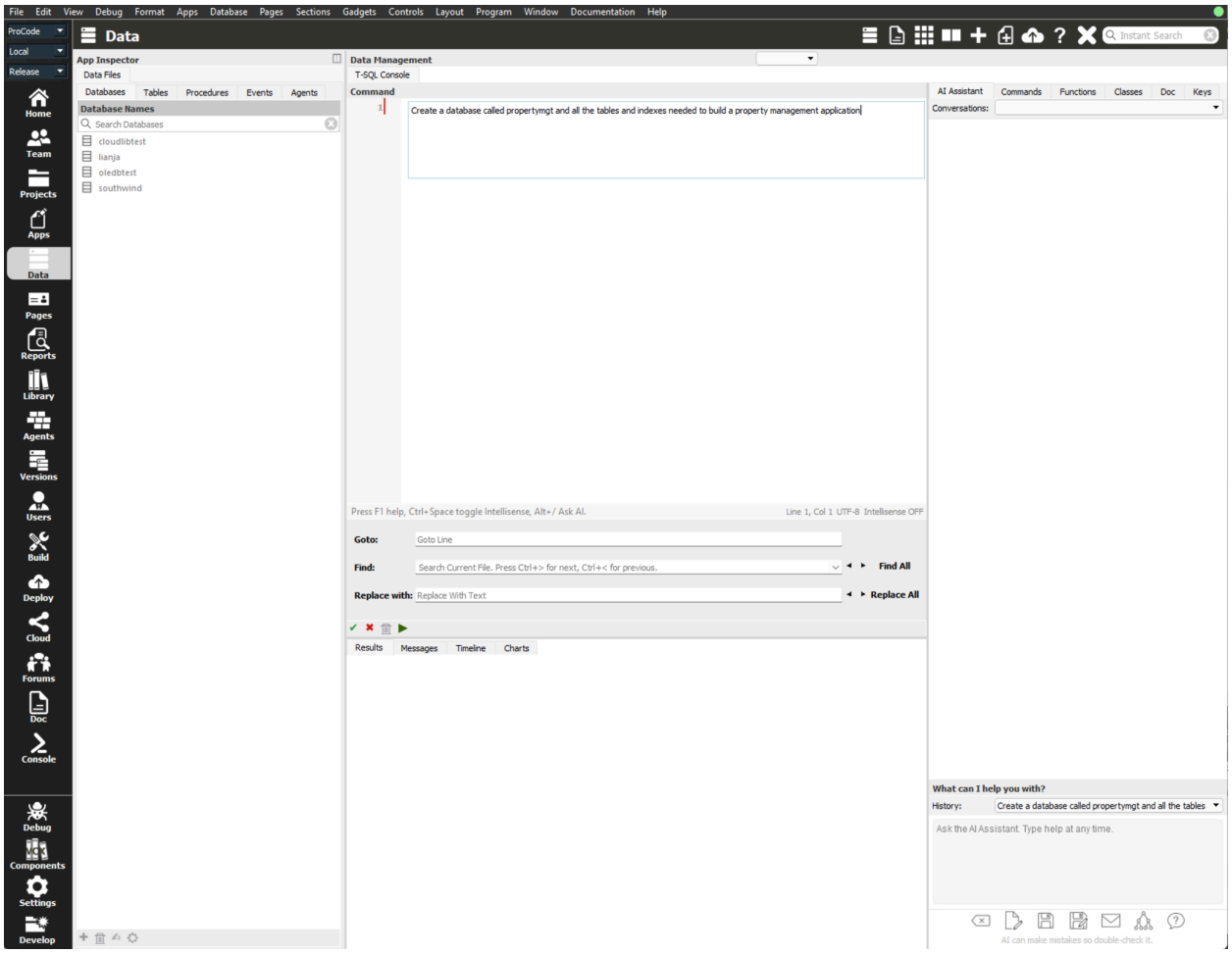


Working in the Data Workspace

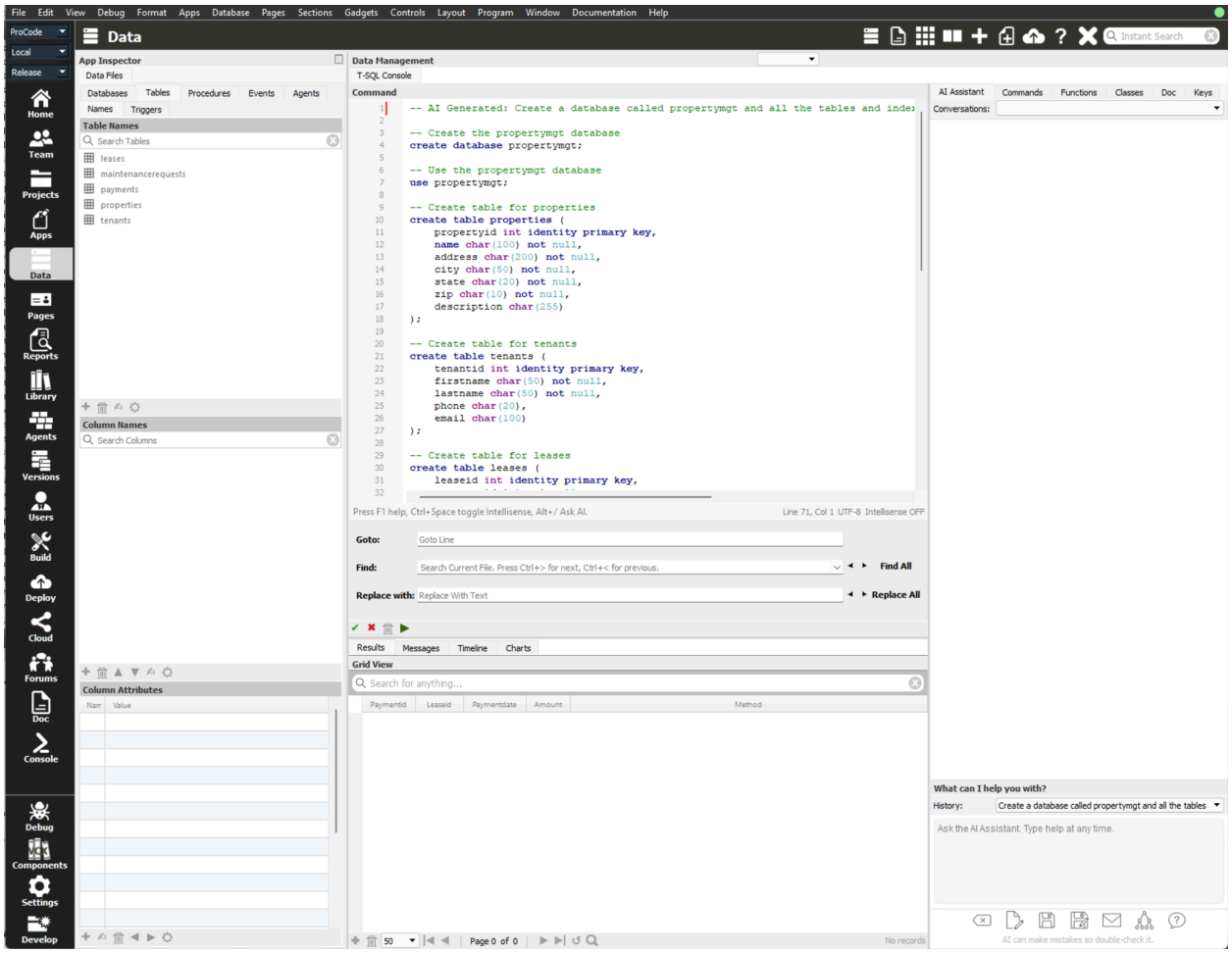
In the T-SQL console.

Press Alt+/ then type:

```
Unset
Create a database called propertymgt and all the tables and indexes needed to build a property management application
```



Press Return and the T-SQL code will be generated for you.



Click the “Execute” icon ▶ and the T-SQL file will be executed.

This is the T-SQL code that was generated then executed to create the database and its tables::

```sql
-- AI Generated: Create a database called propertymgt and all the tables and indexes needed to build a property management application

-- Create the database
create database propertymgt;

-- Use the database
use propertymgt;

-- Create table for properties
create table properties (
  propertyid int identity primary key,
  name char(100) not null,
  address char(200) not null,
  city char(50) not null,
  state char(20) not null,
```

```
  zip char(10) not null,
  description char(255)
);

-- Create index on city for faster searches
create index city on properties(city);

-- Create table for units within properties
create table units (
  unitid int identity primary key,
  propertyid int not null,
  unitnumber char(20) not null,
  bedrooms int not null,
  bathrooms int not null,
  squarefeet int,
  rent decimal(10,2) not null,
  status char(20) not null, -- e.g. Available, Occupied
  constraint fk_units_properties foreign key (propertyid) references properties(propertyid)
);

-- Create index on propertyid for units
create index propertyid on units(propertyid);

-- Create table for tenants
create table tenants (
  tenantid int identity primary key,
  firstname char(50) not null,
  lastname char(50) not null,
  phone char(20),
  email char(100),
  dateofbirth date
);

-- Create index on lastname for tenants
create index lastname on tenants(lastname);

-- Create table for leases
create table leases (
  leaseid int identity primary key,
  unitid int not null,
  tenantid int not null,
  lease_start date not null,
  lease_end date not null,
  rent decimal(10,2) not null,
  securitydeposit decimal(10,2),
  constraint fk_leases_units foreign key (unitid) references units(unitid),
  constraint fk_leases_tenants foreign key (tenantid) references tenants(tenantid)
);

-- Create index on unitid for leases
create index unitid on leases(unitid);

-- Create table for maintenance requests
create table maintenancerequests (
  requestid int identity primary key,
  unitid int not null,
  requestdate date not null,
```

```
  description char(255) not null,
  status char(20) not null, -- e.g. Open, In Progress, Closed
  completiondate date,
  constraint fk_maintenancerequests_units foreign key (unitid) references units(unitid)
);

-- Create index on status for maintenance requests
create index status on maintenancerequests(status);

-- Create table for payments
create table payments (
  paymentid int identity primary key,
  leaseid int not null,
  paymentdate date not null,
  amount decimal(10,2) not null,
  paymentmethod char(50), -- e.g. Credit Card, Check, Cash
  constraint fk_payments_leases foreign key (leaseid) references leases(leaseid)
);

-- Create index on leaseid for payments
create index leaseid on payments(leaseid);
```

AI Assistant deep dive

You chat with the AI Assistant which uses the OpenAI API (Powering ChatGPT) to perform CRUD operations on your *local* private data. This data is under strict security controls based on the logged in users' Roles and Permissions. The AI Assistant is context aware and operates in the following workspaces; Pages, Apps, Library, Data and Agents.

Pages AI Assistant

The AI Assistant in the Pages workspace can be used to query your data using Natural Language Queries at both development and runtime.

As can be seen from the screenshot below, the AI Assistant is displayed on the right of the window.

Clicking the "Undock" icon on the top right of the AI Assistant will undock it into its own window making more space available for UI design if required.

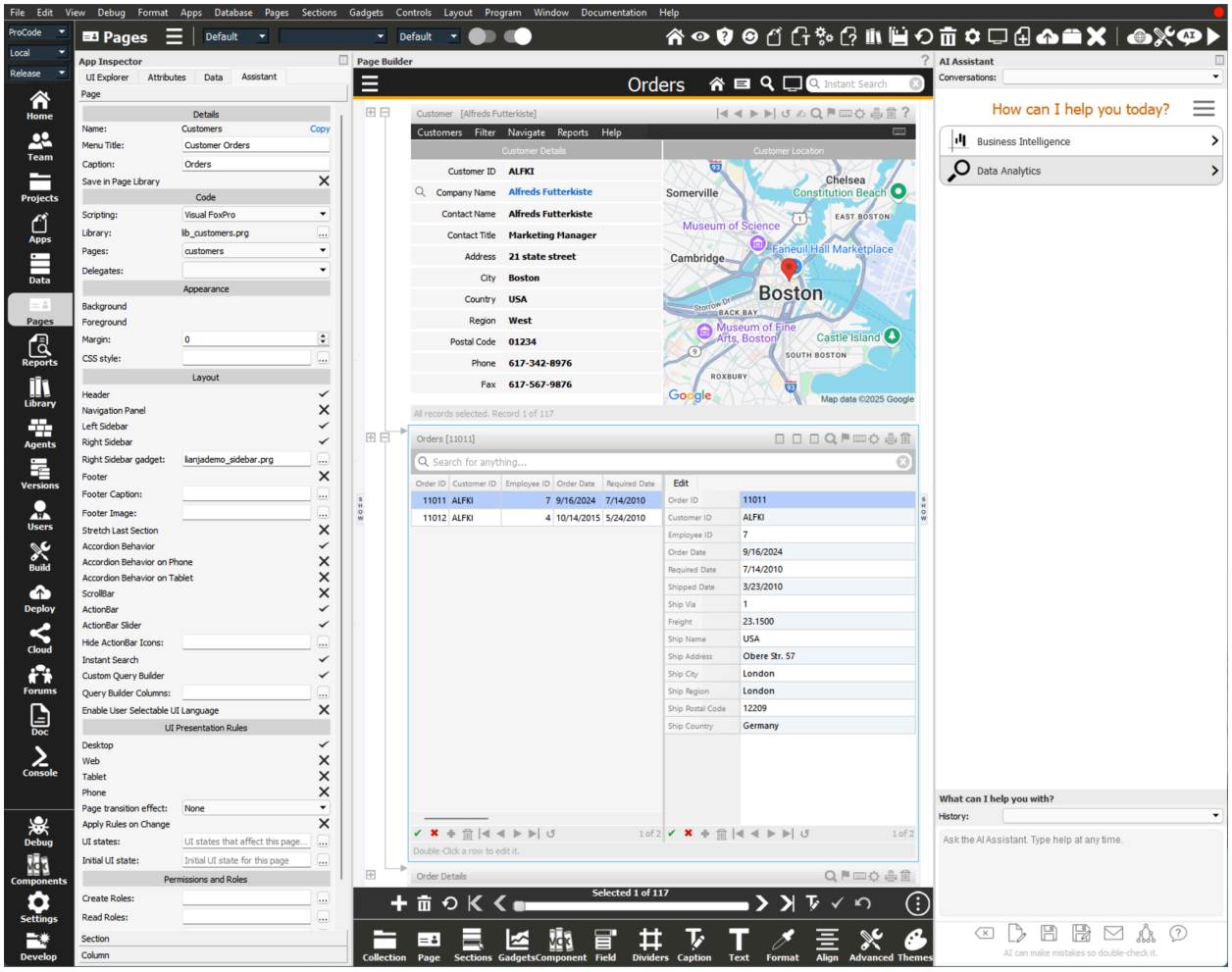
The AI Assistant in the Pages workspace can be toggled on and off by clicking the "AI" icon in the top right of the page.

If you want to disable it completely from either development or runtime mode you can do this in the "App" attributes in the "App Inspector".

The AI Assistant layout

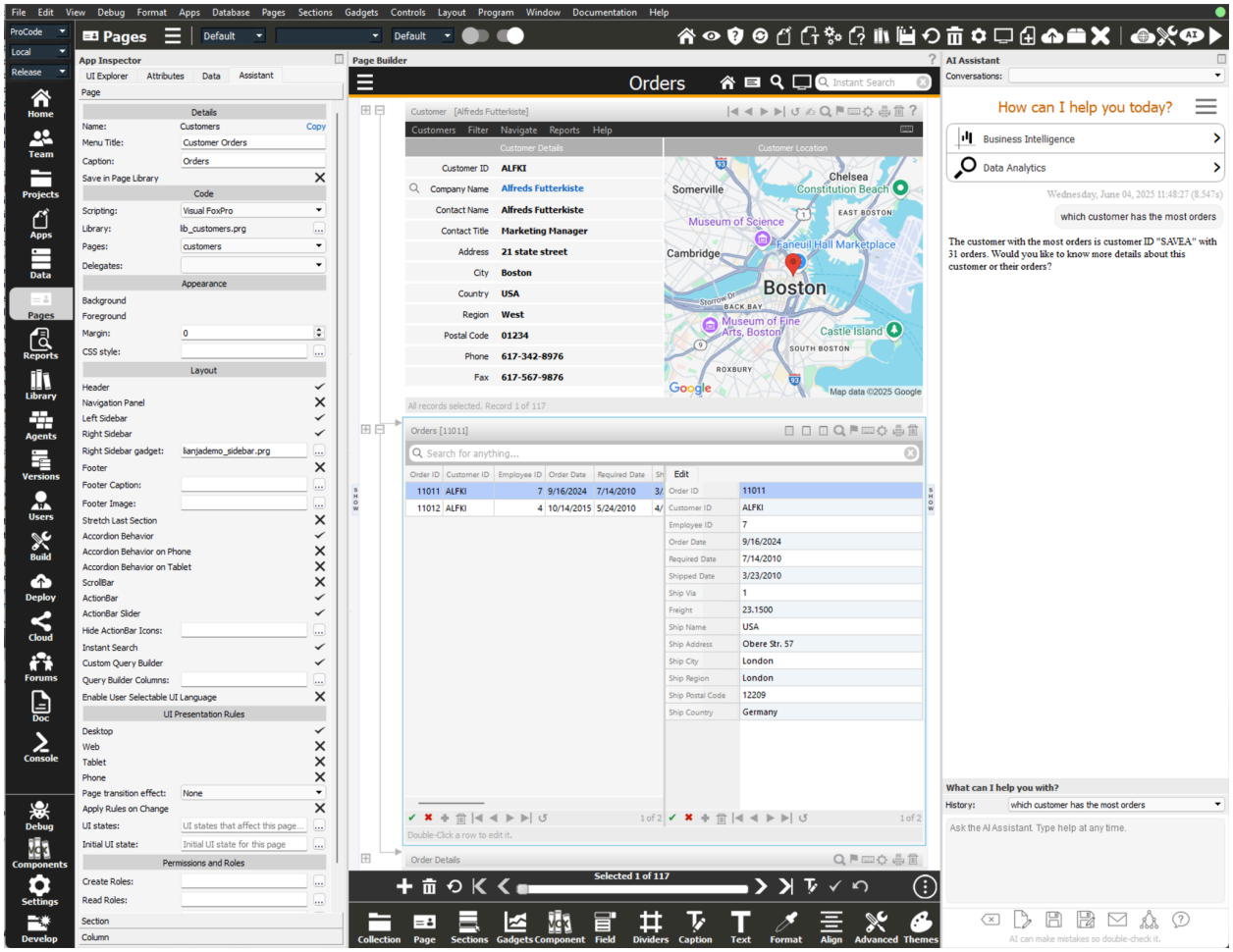
The AI Assistant is laid out from top to bottom like this:

- Header
- Conversations combobox
- Suggestions menu
- Output panel
- Prompt history combobox
- Input panel
- Toolbar
- Progress Bar



Ask AI anything

With an App open and its database open you can “Ask AI” anything about the data without the need to custom code it in the App.

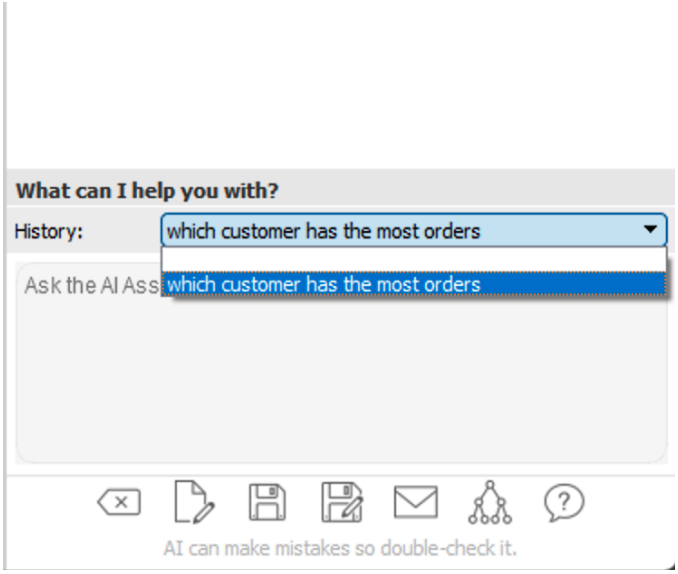


AI Conversations

As you chat with your data, you can save your current chat history as a “conversation” with a name. This becomes the equivalent of an App. You can replay conversations which replays all of the chat in a conversation. This is a further abstraction on NoCode App building.

Conversation History

As you ask questions (known as prompts) the history of your questions is maintained in the history combobox. You can select previous prompts and replay them just by selecting them.

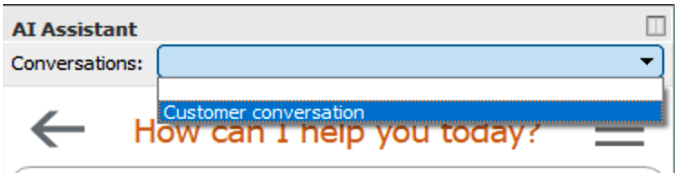


Saving and Replaying Conversations

To save a conversation click the “Save Conversation” icon  in the AI Assistant toolbar.

Choose a name for the conversation.

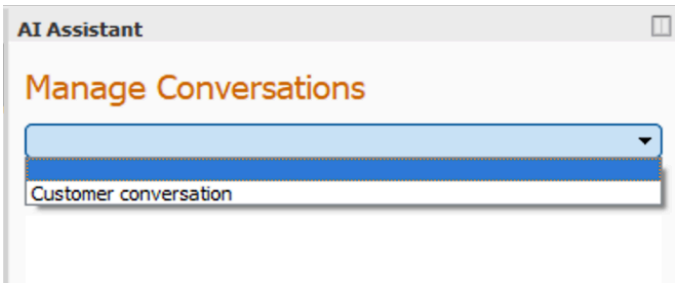
This conversation will persist across Lianja sessions and can be replayed just by selecting it from the Conversations combobox.



Managing Conversations

You can manage previously saved conversations.

Click the “Manage Conversations” icon  in the AI Assistant toolbar.



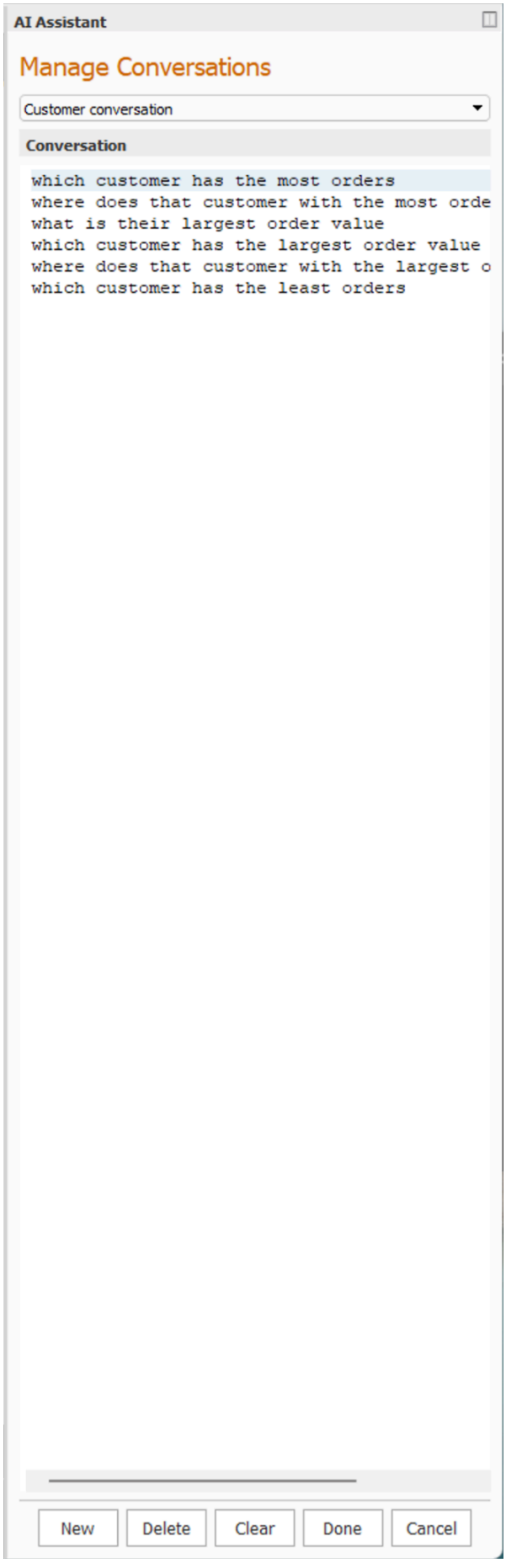
Choose a conversation from the combobox at the top and you can edit it, add additional “prompts” or correct prompts.

The text that you want displayed into the output panel of the AI assistant can be different to the prompt that is asked. To accomplish this just postfix the prompt with a # followed by the text you want displayed.

Prompts can include {macros} so that they can be asked in the context of the current “Data View”.

e.g.

```
Unset
Show me the orders for {customers.customerid}.
```


AI Assistant
Manage Conversations
Customer conversation
Conversation
which customer has the most orders
where does that customer with the most orde
what is their largest order value
which customer has the largest order value
where does that customer with the largest o
which customer has the least orders
New
Delete
Clear
Done
Cancel

Email Conversations

At any time during a conversation you can email the conversation to others.

Click the “Email” icon  in the AI Assistant toolbar.

The “Compose Email” panel will overlay on top of the AI Assistant.

You can specify multiple email recipients by separating the email addresses by commas.

If you uncheck the “Heading” checkbox, then the standard heading will not be included in the email.

You can edit the “Message” also.

Click “Send” to send the email to one or more recipients and return to normal AI Assistant operation.

AI Assistant

Compose Email

To

Cc

Subject Conversation History Report

Heading

Message

Please find attached a Conversation History Report.

Best Regards

admin

Conversation History

Monday, June 09, 2025 14:05:29 (2.865s)

Average order value

The average order value for the sales database is approximately $1,635.43. If you need any more details or further analysis, please let me know!

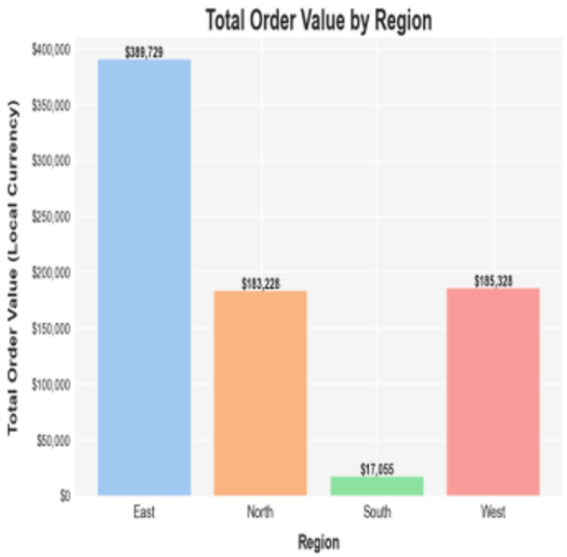
Monday, June 09, 2025 14:05:33 (2.891s)

Total number of orders

There are a total of 820 orders in the orders table. If you need any additional details or analysis, please let me know!

Monday, June 09, 2025 14:06:18 (37.396s)

Sales by region



Cancel Send

Understanding Conversation Context

As you ask questions in the AI Assistant the questions and their answers are remembered until you click the ⌫ “Clear” icon.

The “Conversations Context” is maintained until you clear it. As you ask questions, they are based on the context of previous questions and their answers.

e.g

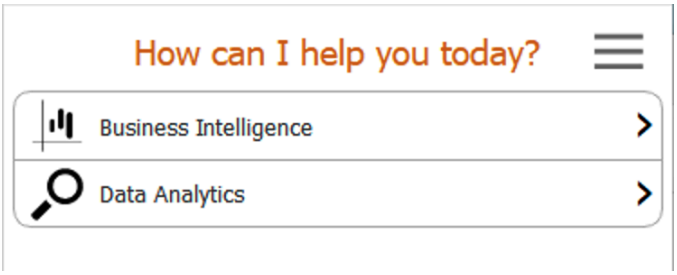
```
Unset
Which customer has the most orders?
Where do they live?
```

Without conversation context the second question would make no sense, but when asked in this context the correct address will be displayed.

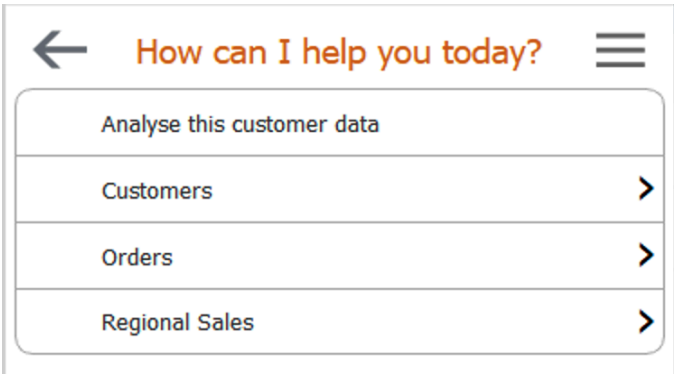
Suggestion Menus

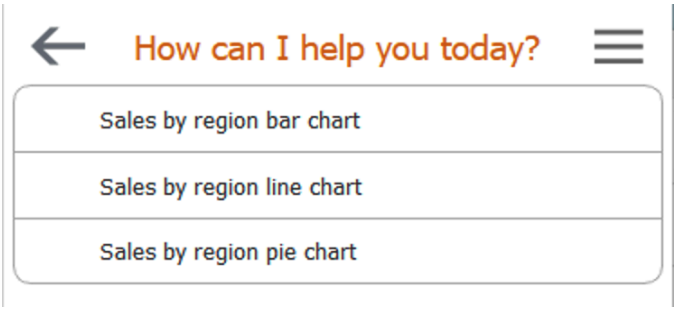
Suggestion menus provide a hierarchical menu driven interface for users to navigate custom defined questions and actions.

e.g.



Suggestions that have a **>** on the right have submenus which you can click to navigate down through the menus.



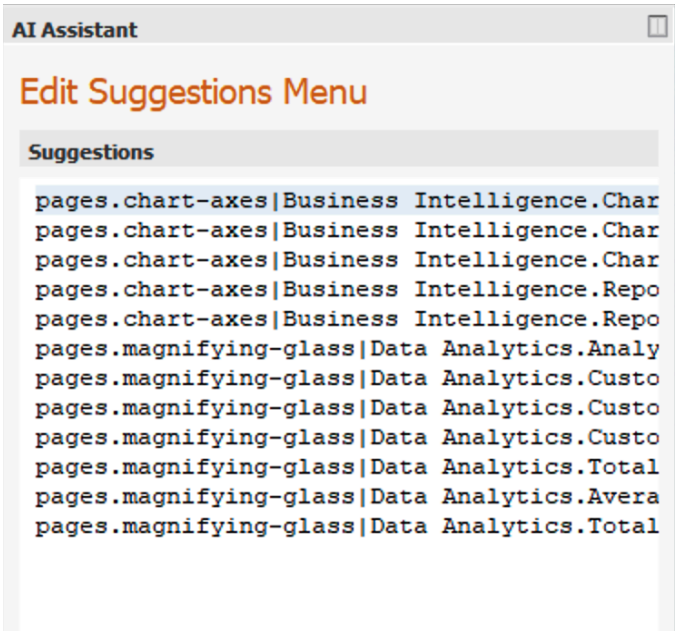


Click the ← "Back" button to navigate backwards to the previous menu.

Click the ☰ "Menu" button to toggle between hiding and showing the suggestions menu.

Editing Suggestion Menus

You create and/or edit suggestions menus by clicking the "Edit Suggestions Menu" icon.



The format of the suggestions are as follows:

pages.chart-axes|Business Intelligence.Charts.Plot Bar Chart=\
Plot a bar chart for 240,360,120,480|140,340,80,200 title Sample Bar Chart##Plot a bar chart

Note that each line in the file can use \ as a line continuation character.

Optional. only display for the specified database
[database]

Workspace this suggestion belongs to with an optional icon. Icons come from lib:/icons (pages, data, apps, library or agents)
pages.chart-axes

Dotted path of menu item
Business Intelligence.Charts.Plot Bar Chart

The prompt to be sent to the AI Provider. The text following the optional ## is the text to be displayed alongside the AI result in the output panel.
Plot a bar chart for 240,360,120,480|140,340,80,200 title Sample Bar Chart##Plot a bar chart

The example suggestions shipped contain:

```
Unset
pages.chart-axes|Business Intelligence.Charts.Plot Bar Chart=\
Plot a bar chart for 240,360,120,480|140,340,80,200 title Sample Bar
Chart##Plot a bar chart
pages.chart-axes|Business Intelligence.Charts.Plot Pie Chart=\
Plot a pie chart for 240,360,120,480 title Sample Pie Chart##Plot a pie chart
pages.chart-axes|Business Intelligence.Charts.Plot Line Chart=\
Plot a line chart for 240,360,120,480140,340,80,200 title Sample Line
Chart##Plot a line chart
pages.chart-axes|Business Intelligence.Reports.Order Summary=\
show report order summary
pages.chart-axes|Business Intelligence.Reports.Customer Summary=\
show report customers and their orders
pages.magnifying-glass|Data Analytics.Analyse this Dataview=\
Analyse this Dataview
pages.magnifying-glass|Data Analytics.Customers.Customer with most orders=\
which customer has the most orders
pages.magnifying-glass|Data Analytics.Customers.Customer with the largest order
value=\
which customer has the largest order value
pages.magnifying-glass|Data Analytics.Customers.Customer has the least orders=\
which customer has the least orders
pages.magnifying-glass|Data Analytics.Total orders=\
How many orders are there
```

```
pages.magnifying-glass|Data Analytics.Average orders=\
What is the average order value
pages.magnifying-glass|Data Analytics.Total order value=\
What is the total order value
```

Data AI Assistant

The AI Assistant in the “Data” workspace operates differently than that in the “Pages” workspace.

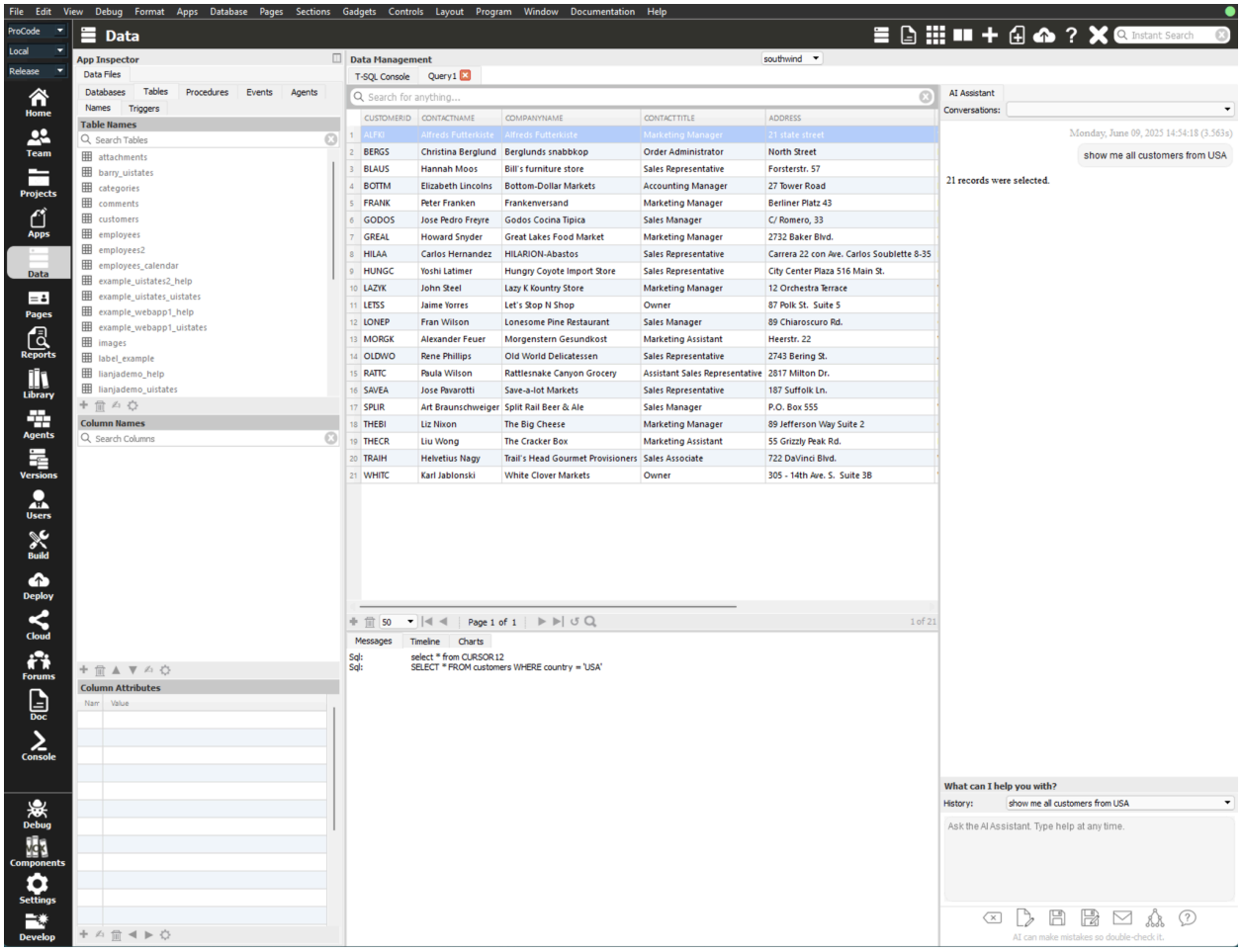
AI Powered Business Intelligence

For any prompts (questions) asked in the AI Assistant, it knows about the current database that is open and its tables and columns and will run SQL queries and display the data selected in a grid.

e.g.

```
Unset

Show me all customers from usa
```



Once data is displayed in a grid you can drill down into the data using Natural Language Queries.

e.g. in the input panel type:

```
Unset

Filter containing usa
```

Or alternatively type in the grid “Search Bar”.

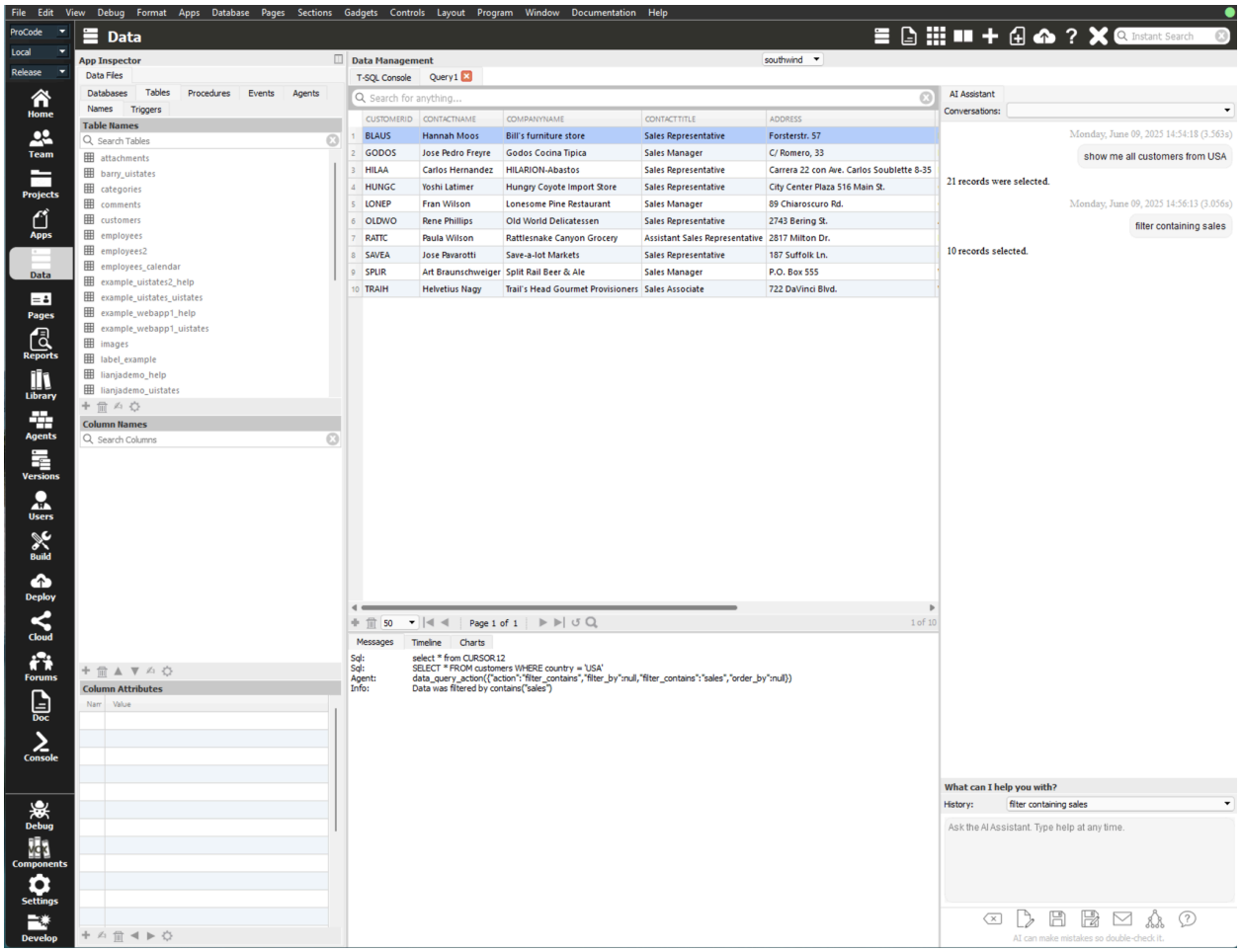
```
Unset
usa
```

The data in the grid will be filtered and the grid will be re-displayed.

You can continue filtering the data with other filter prompts which are additive to the existing filters.

To reset the filter in the input panel type:

```
Unset
Clear filter
```



Notice how the “Messages” tab at the bottom shows you the SQL being executed.

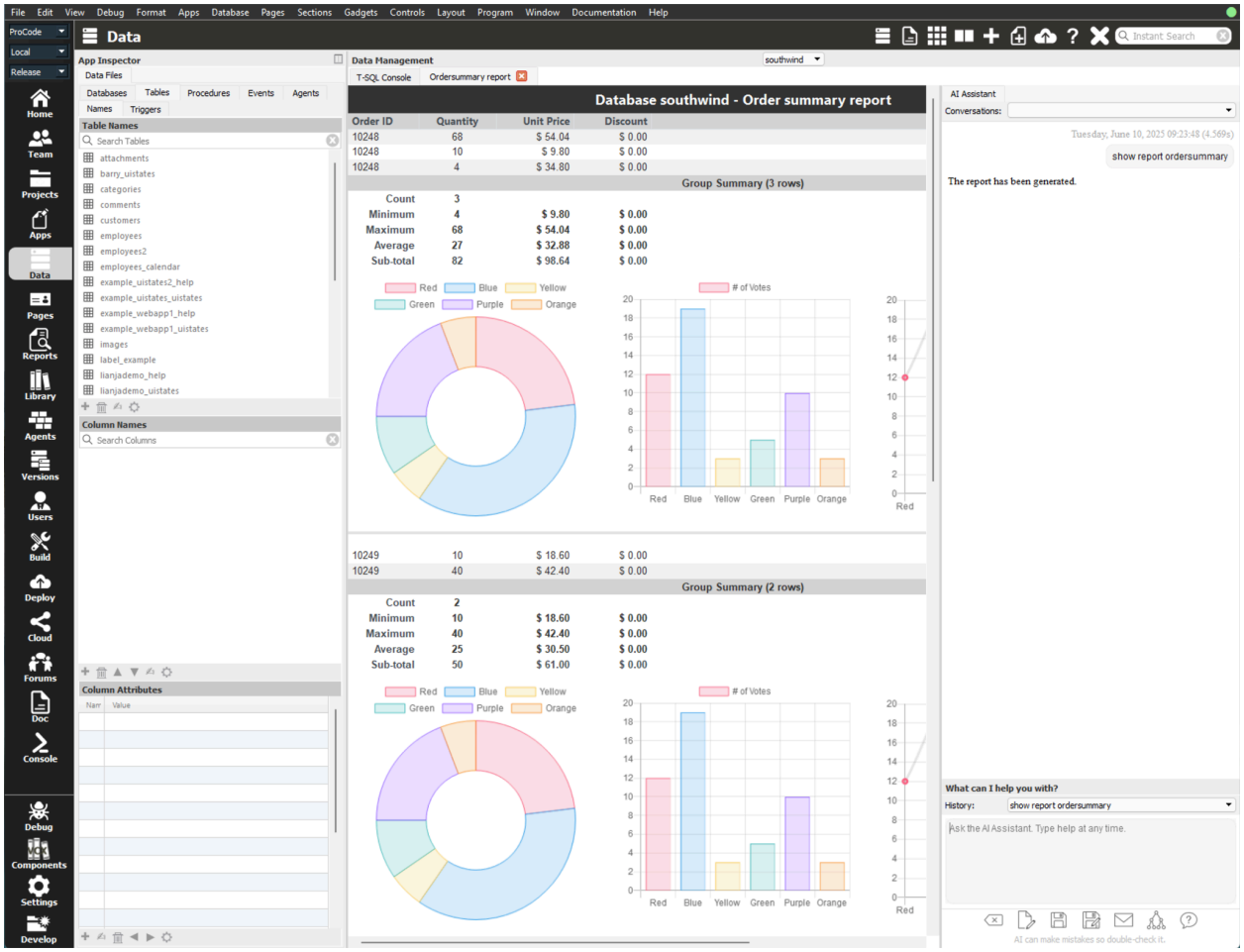
Business Intelligence Reporting

You can generate reports in the “Data” workspace which are displayed in a tab alongside your queries and T-SQL stored procedures. You create reports in the “Reports” workspace using the “Report Builder”.

In the AI Assistant input panel type:

```
Unset
Show report ordersummary
```

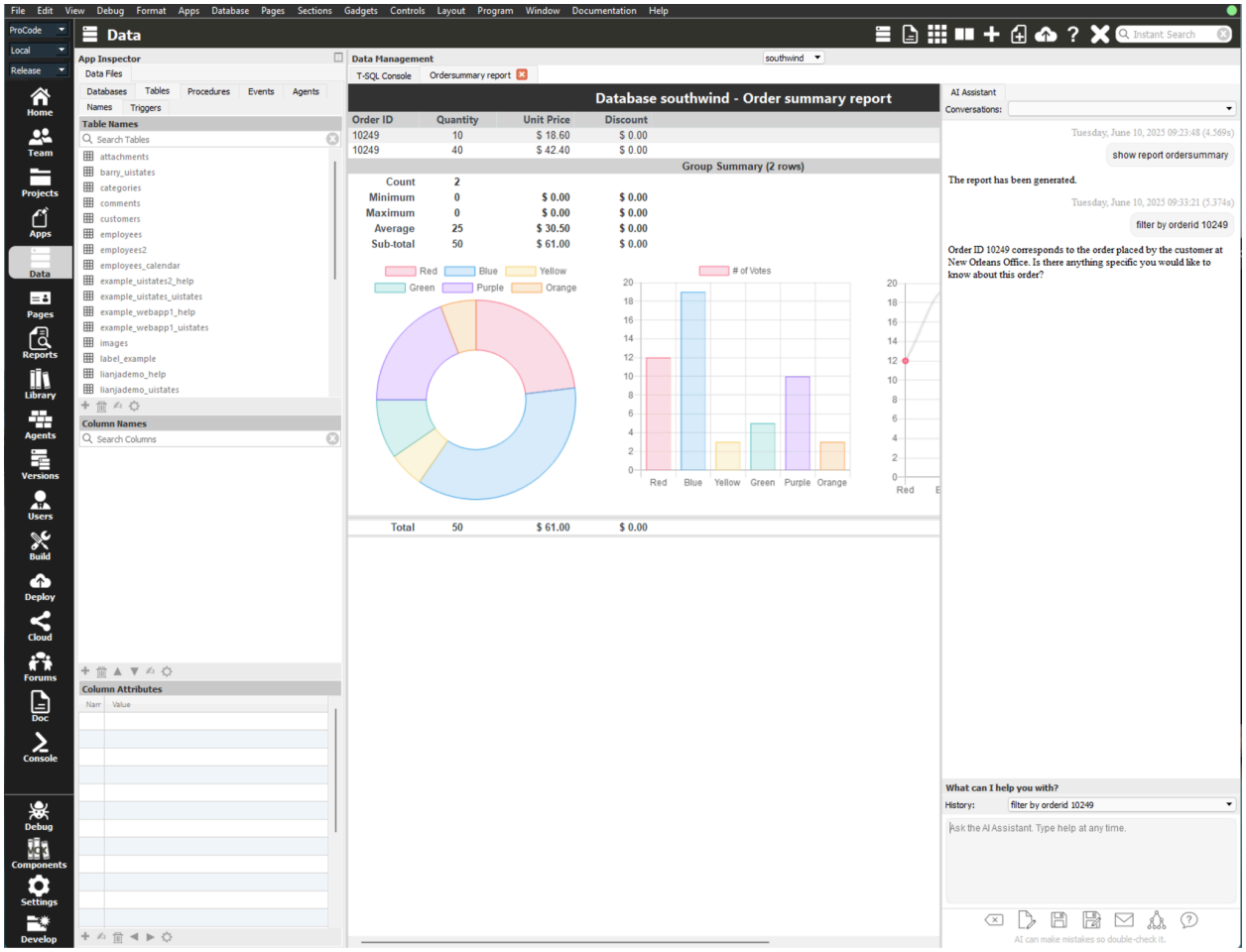
And the report will be displayed.



With the tab of the report selected, in the AI Assistant input panel type:

```
Unset
Filter by orderid 10249
```

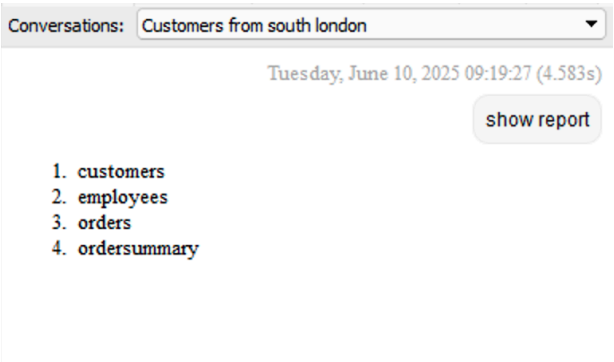
The report will be dynamically filtered and redisplayed.



If you want to see a list of available reports and choose one, type:

```
Unset
Show report
```

A list of available reports will be displayed in the output panel.



Choose an option by typing (in this case) a number from 1-4 and the report will be generated and displayed.

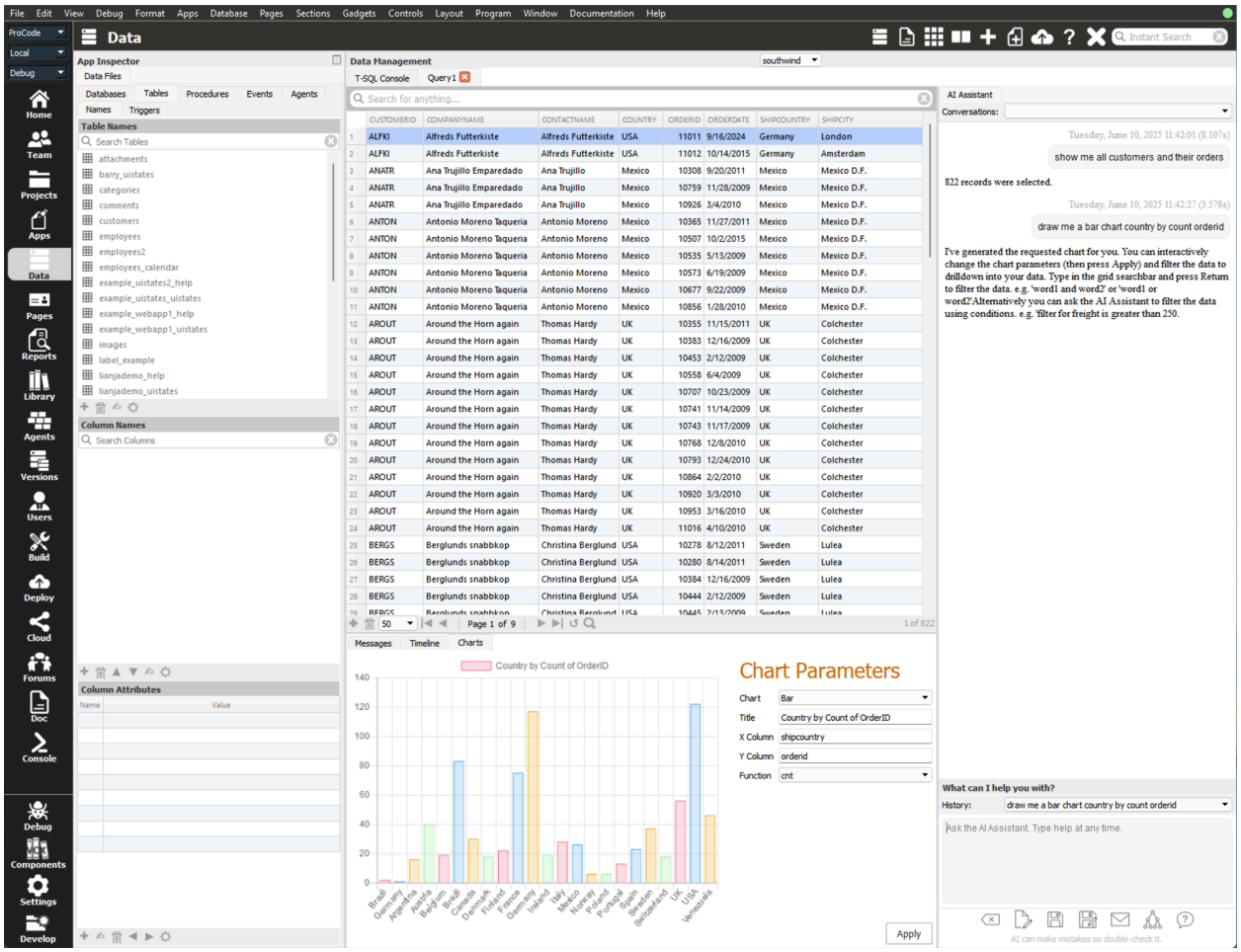
Data Visualization

You can also visualize your data as charts in the “Data” workspace. These are displayed in the “Charts” tab at the bottom.

In the AI Assistant input panel type:

```
Unset
show me all customers and their orders
draw me a chart country by count orders
```

And the chart will be displayed.



The charts in the “Data” workspace are interactive so you can change the Chart Parameters and click “Apply” to view your data in a variety of ways; Bar, Line, Pie.

You can filter the data in the AI Assistant:

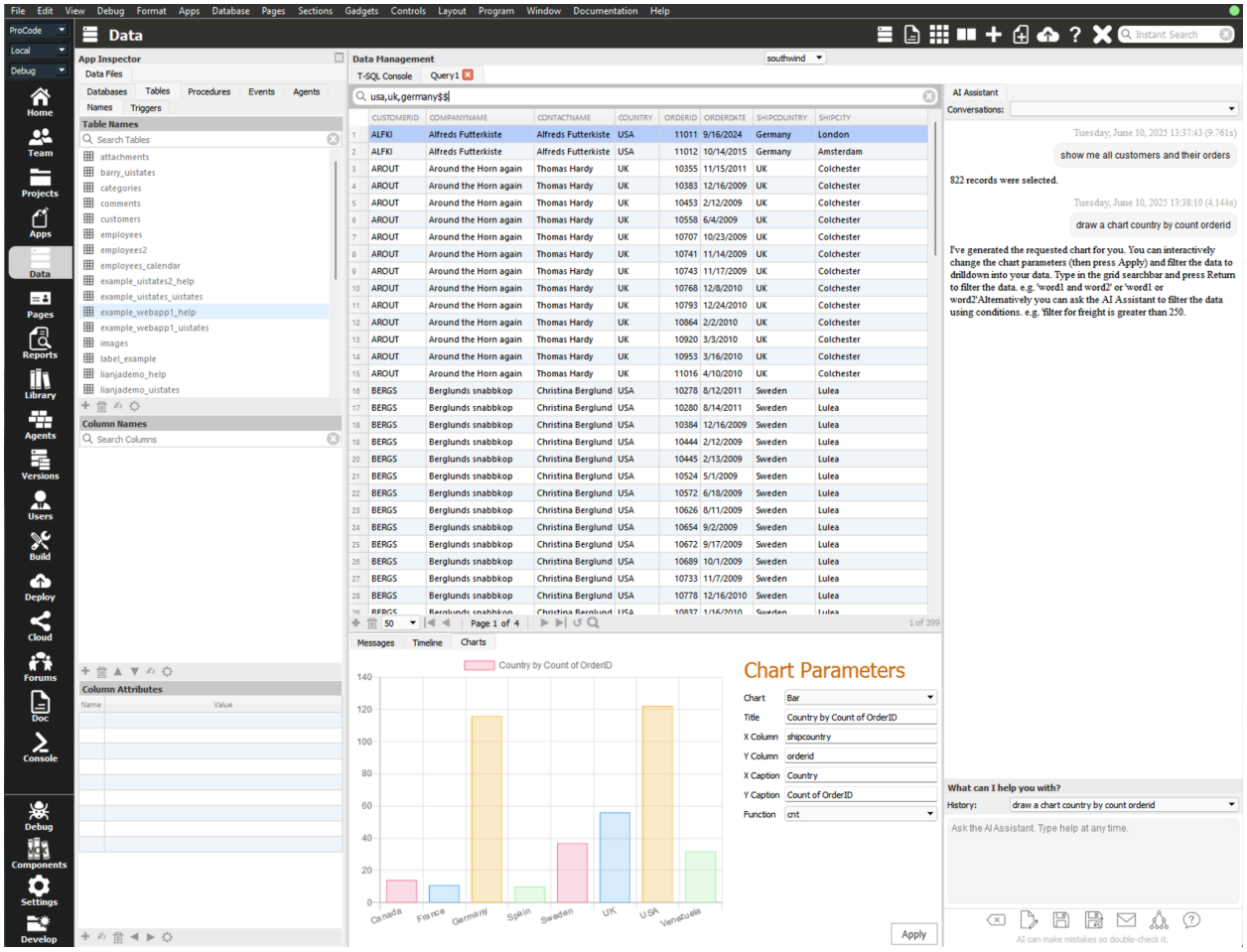
e.g.

```
Unset
Filter containing "usa,uk,germany"
```

Or alternatively by typing in the grid “Search Bar”:

```
Unset
usa,uk,germany
```

The data will be filtered and the chart will be re-displayed as the data in the grid is refreshed.

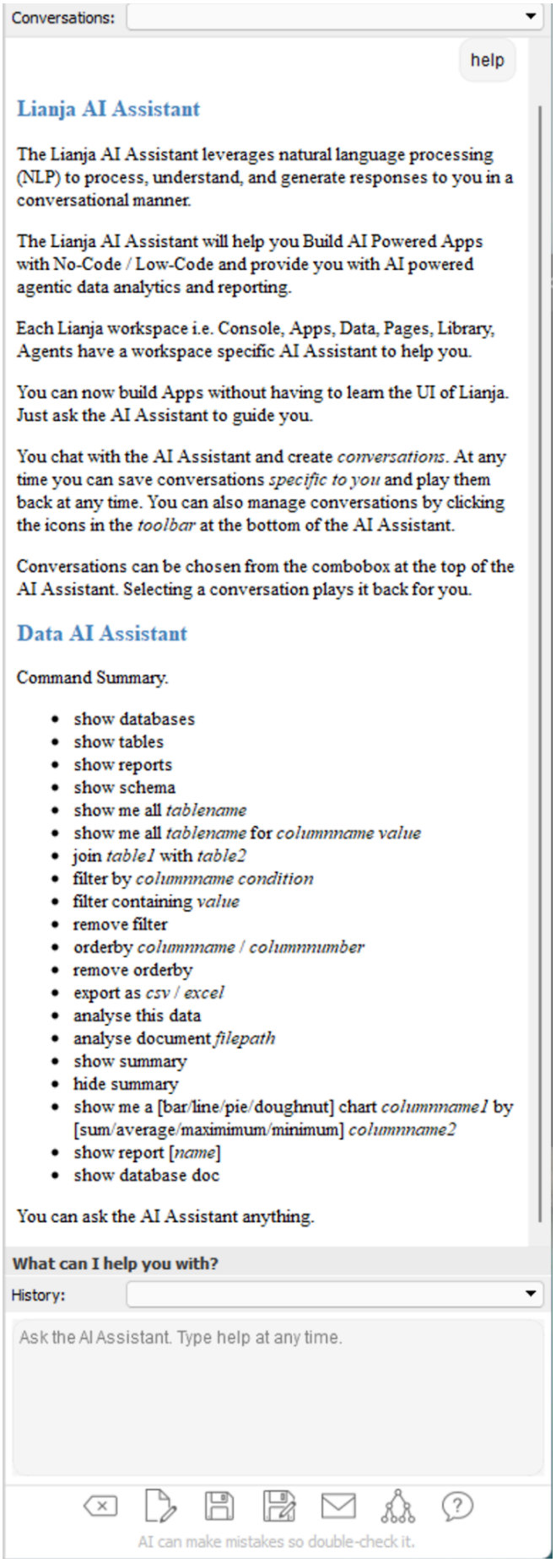


What can you do in the Data workspace

At any time in the AI Assistant you can type:

```
help
```

And a list of built-in shortcut commands will be displayed specific to the specific workspace that you are currently working in. In the case of the “Data” workspace:


Conversations:
help
Lianja AI Assistant
The Lianja AI Assistant leverages natural language processing (NLP) to process, understand, and generate responses to you in a conversational manner.
The Lianja AI Assistant will help you Build AI Powered Apps with No-Code / Low-Code and provide you with AI powered agentic data analytics and reporting.
Each Lianja workspace i.e. Console, Apps, Data, Pages, Library, Agents have a workspace specific AI Assistant to help you.
You can now build Apps without having to learn the UI of Lianja. Just ask the AI Assistant to guide you.
You chat with the AI Assistant and create conversations. At any time you can save conversations specific to you and play them back at any time. You can also manage conversations by clicking the icons in the toolbar at the bottom of the AI Assistant.
Conversations can be chosen from the combobox at the top of the AI Assistant. Selecting a conversation plays it back for you.
Data AI Assistant
Command Summary.
show databases
show tables
show reports
show schema
show me all tablename
show me all tablename for columnname value
join table1 with table2
filter by columnname condition
filter containing value
remove filter
orderby columnname / columnnumber
remove orderby
export as csv / excel
analyse this data
analyse document filepath
show summary
hide summary
show me a [bar/line/pie/doughnut] chart columnname1 by [sum/average/maximimum/minimum] columnname2
show report [name]
show database doc
You can ask the AI Assistant anything.
What can I help you with?
History:
Ask the AI Assistant. Type help at any time.
AI can make mistakes so double-check it.

Building custom AI Agents

Creation and editing of custom AI Agents can be done with minimum to no coding whatsoever just by asking in the AI Assistant. e.g

```
Unset
Create an agent in Python to get the current weather?
```

```
Unset

Create an agent in Python to send email using sendgrid?
```

The AI Assistant is multilingual providing support for many languages; English, French, German, Spanish, Portuguese etc. Questions asked will be answered in the same language as they were asked in.

Talk to any Lianja accessible data in real time; native Lianja, MSSQL, MySQL etc! Visualize it in forms, grids, charts, reports and other custom formats. Save it and optionally email the results in a wide variety of formats including excel, csv, xml and custom formats such as invoices!

Let's build a custom AI Agent

Select the "Agents" workspace.

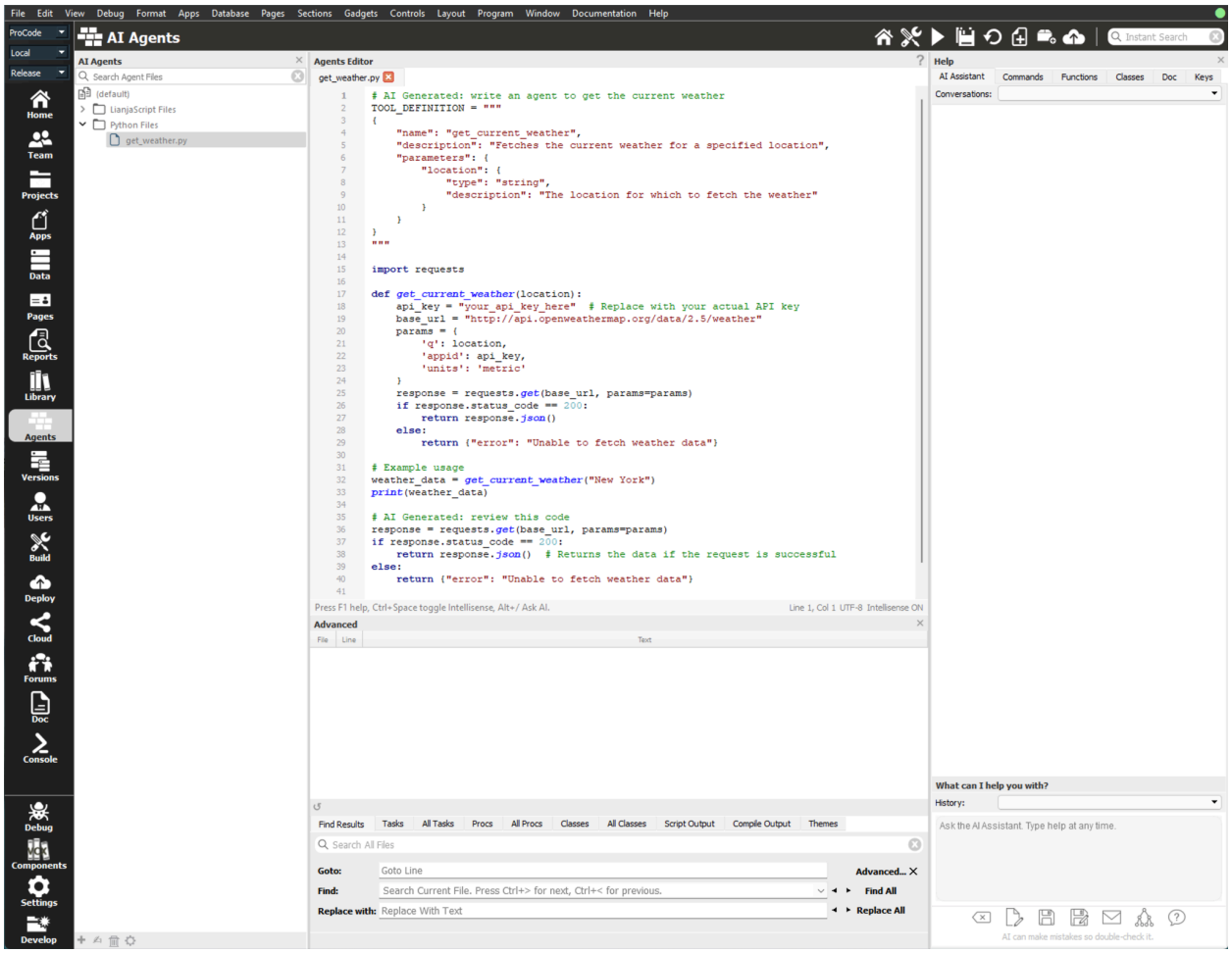
Custom AI agents can be written in either LianjaScript or Python.

For this example, create a "New" Python file called get_weather.py.

Press Alt+/ then type:

```
Unset
Create an agent to get the current weather. Do not add the code for an example
call of the agent function?
```

The Python code will be generated for the agent with the corresponding "TOOL_DEFINITION" needed by OpenAI.



The Python code generated:

```python
# AI Generated: write an agent to get the current weather
TOOL_DEFINITION = """
{
  "name": "get_current_weather",
  "description": "Fetches the current weather for a specified location",
  "parameters": {
    "location": {
      "type": "string",
      "description": "The location for which to fetch the weather"
    }
  }
}
"""
import requests
def get_current_weather(location):
  api_key = "your_api_key_here"  # Replace with your actual API key
  base_url = "http://api.openweathermap.org/data/2.5/weather"
  params = {
    'q': location,
```

```
    'appid': api_key,
    'units': 'metric'
}
response = requests.get(base_url, params=params)
if response.status_code == 200:
    return response.json()
else:
    return {"error": "Unable to fetch weather data"}
```

Add the name of the “Agent” to the *agents.conf* file in the “agents” workspace in the format below. Notice how I have prefixed the filename with a ‘-’ sign. This prevents the agent from being used in the AI Assistant *other than* in the “Agents” workspace. After testing you should remove the ‘-’ sign so that the agent is exposed and used by the AI Assistant in all workspaces.

File: agents.conf

```
Unset
-get_weather.py=get_current_weather(location)
```

Format:

get_weather.py	Name of the file containing the agent code
get_current_weather(args…)	The function in the file to handle the agent call

Note that the name **get_current_weather** corresponds to the **name** in the TOOL_DEFINITION. and **args** correspond to the parameters in the TOOL_DEFINITION.

Installing the agent

To test the agent with the AI Assistant you first need to “Install” it.

In the “Agents” workspace HeaderBar, click the “Install Agent” (second from the right).

Testing the agent

Staying in the “Agents” workspace ask the AI Assistant:

```
Unset
What is the current weather in Boston?
```

If you have an api_key in the **get_current_weather()** function your agent will retrieve the current weather in Boston and display it in the AI Assistant output panel.

Publishing Agents into the Agents repository

Agents can be uploaded into the Agents repo in github. This makes them discoverable by others so that developers can share agents.

Before uploading to github you need to create a manifest file for the agent describing it. This information is used when searching for Agents in the “Agents” workspace.

This manifest file has the same name as the agent file but with a .json extension. E.g in this case *get_weather.json*.

This manifest file has the following JSON format.

```JSON
{
        "name":"get_weather",
        "displayname":"get_weather",
        "description":"Fetches the current weather for a specified location",
        "publisher":"Your name",
        "version":"1.0",
        "tags":"#agent,#weather,#python"
}
```

The “tags” are used to simplify agent search discoverability.

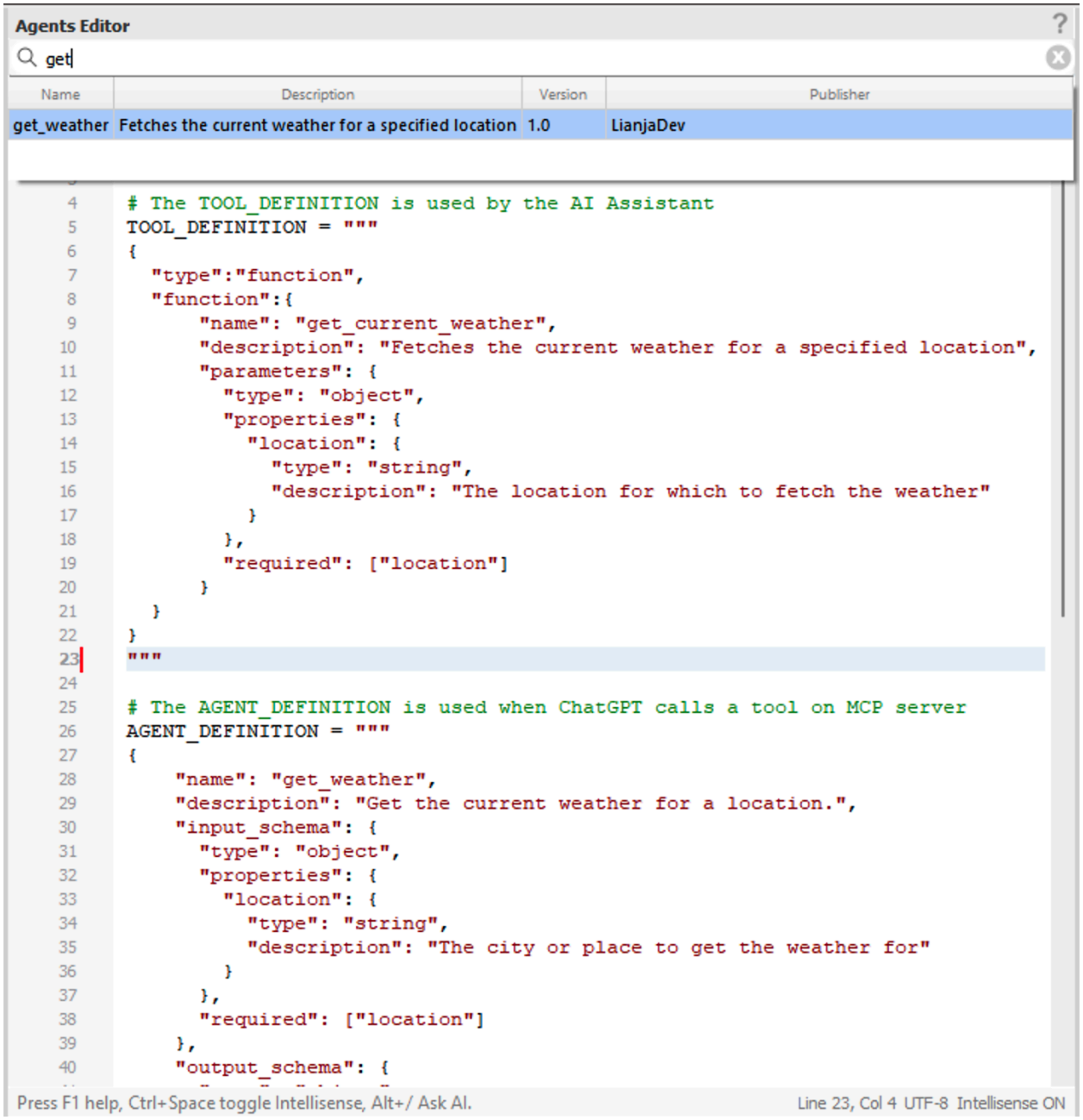
Before uploading you should fill in the Agents Repo Attributes in the App Attributes; including the Personal Access Token assigned to you from the Lianja Dev Team. If you want to publish agents you will need this so contact us at support@lianja.com. You can of course optionally create your own github account and then create an agents repo private to your company.

Agent discovery and installation

Agents can be shared by developers by publishing them as described above.

You can search for agents in the Agents repo directly from the editor in the “Agents” workspace. As you type in the agents SearchBar any agents that have #tags or descriptions containing the text you type will be displayed.

Choose an agent by double clicking it in the grid or pressing <return>, and it will be downloaded and installed for you.



After installation of an agent, it will NOT be active on your system until you add it to *agents.conf* as described previously.

Deploying agents into the Cloud

You deploy agents into the cloud by creating a Lianja package in the “Deploy” workspace of the Lianja App Builder.

After creating a package use the “Lianja Admin Console” on your Lianja Cloud Server instance (or tenancy in the Lianja Cloud) to upload and install the package.

Limitations and Restrictions

The Lianja 11 App Builder has built-in support for the OpenAI API.

In order to minimize unfair use of the AI capabilities of Lianja 11, there are limits on built-in AI token usage. This is currently set at 100,000 per hour. If you need more than this please contact sales@lianja.com.

AI providers such as OpenAI charge for input and output token use.

If you are using the Lianja AI Assistant built-in AI you will be restricted to 100,000 tokens per hour.

There is a progress bar at the very bottom of the AI Assistant which shows you your current usage (hover the mouse over it for details). This resets every hour.

If you want to use more than 100,000 tokens per hour you need to subscribe to the OpenAI API yourself and obtain an API key. This API key should be specified in an environment variable OPENAI_API_KEY for all your users, or alternatively specified in the App attributes using the App Inspector.

Note that runtime deployed Apps require you to have your own private OpenAI API key.