Data specification navigator  
(user documentation)

# Introduction

This document is a user guide for the Data specification navigator application. It is intended for end-users who wish to navigate and query data specifications using simple, natural language.

The Data Specification Navigator is a conversational tool designed to help you interact with your data specification. It allows you to ask questions about your data specification in plain English and, in return, provides you with the correct technical query (SPARQL) to get the data you need.

## Target audience

This document's target audience are end-users who want to start using the application. It is assumed that the application has been deployed and is accessible to the user.

# Getting started

This application works directly with data specifications created in the Dataspecer tool therefore, some knowledge about Dataspecer is required. If you don’t know how to use Dataspecer, please consult its documentation and guides: <https://dataspecer.com/>

## Workflow

This section demonstrates the workflow with this application on a concrete example with screenshots for better visualization. The important UI elements are discussed in more detail in chapter 3).

The user navigates to the main page of the Data specification navigator, which is by default the conversation management view (see Figure 1).

Figure 1: No conversations created yet

A screenshot of a chat

AI-generated content may be incorrect.

The user follows the instructions and creates a new Dataspecer package. The user then copies the package IRI (see Figure 2).

Figure 2: Dataspecer package UI

A screenshot of a computer

AI-generated content may be incorrect.

The user returns to the Data specification navigator, clicks on the “Create new conversation” and a form appears. The user fills in the necessary information (see Figure 3).

Figure 3: Conversation creation form

A screenshot of a computer

AI-generated content may be incorrect.

The user opens the newly created conversation and sees the welcome message (see Figure 4**Error! Reference source not found.**).

Figure 4: Welcome message

A screenshot of a chat

AI-generated content may be incorrect.

The user types their first message and sends it (see Figure 5).

Figure 5: Waiting for a chatbot reply

A screenshot of a chat

AI-generated content may be incorrect.

The chatbot sends an answer back. The right-side panel is updated with mapped items from the data specification. The reply from the chatbot is long so it is shown in two parts (see Figure 6 and Figure 7).

Figure 6: Chatbot reply - SPARQL query

A screenshot of a chat

AI-generated content may be incorrect.

Figure 7: Chatbot reply - suggestions

A screenshot of a chat

AI-generated content may be incorrect.

The toggle “Show your message” shows the user message with the highlighted parts corresponding to the mapped items (see Figure 8).

Figure 8: Words highlighting

A screenshot of a chat

AI-generated content may be incorrect.

The user can click on a mapped word in their message, and the UI will display a summary for that item (see Figure 9).

Figure 9: Mapped item summary

A screenshot of a computer

AI-generated content may be incorrect.

The user can click on a suggestion in the chatbot’s reply and the UI will display a summary for that item (see Figure 10).

Figure 10: Suggested item summary

A screenshot of a computer

AI-generated content may be incorrect.

The user selects some suggested items. The UI shows the button “Add all selected items to my message” on top of the chat (see Figure 11).

Figure 11: User chooses suggestions

A screenshot of a computer

AI-generated content may be incorrect.

The user clicks the button to add all items, which prompts the UI to retrieve the suggested message from the backend (see Figure 12).

Figure 12: Waiting for a suggested message

A screenshot of a chat

AI-generated content may be incorrect.

The UI displays the received suggested message (see Figure 13).

Figure 13: Suggested message displayed

A screenshot of a chat

AI-generated content may be incorrect.

The user sends the suggested message as is. Alternatively, the user can modify the message before sending it.

The chatbot replies with the new answer (see Figure 14). Notice that the right-side panel is updated with newly mapped items.

Figure 14: User sends suggested message and receives a reply

A screenshot of a chat

AI-generated content may be incorrect.

The conversation flow can continue in this manner until the user is satisfied.

# Available operations

This section chapter describes the functionalities of the application and its important UI elements.

## Creating a conversation

To start working with the application, the user needs to navigate to the app’s website and create a new conversation. If no conversations have yet been created, the UI will display a short message instructing the user to create a Dataspecer package and then create a new conversation.

To create a new conversation, the user must click on the button “Create new conversation”. In the popup box, the user must fill in the correct IRI for the package they want. The field Dataspecer package name does not have to match the actual package’s name. The user can choose to fill in anything. This name **does not have any influence** on how the application works nor does it affect any packages in Dataspecer. Once all the fields are filled in, the user can click on the “Create conversation” button. After the conversation has been created, the user can open the conversation to start chatting with the chatbot.

## Conversation

The conversation page is split vertically into two main areas. On the left side are the chat messages and the input box, where the user can type their messages. On the right side is a panel that displays data specification items, which the chatbot has managed to map to given the messages from the user. This right-side panel can be hidden by clicking on the “Hide mapped data specification items” button.

The first message in the conversation is always sent by the chatbot and has a green background. It informs the user about how many classes and properties has been found in the data specification. Additionally, it gives the user a brief summary of what the data specification is about and suggests a few things the user might want to ask about.

### Sending a message

The user can type a message into the input box on the bottom of the chatting interface. The user sends their message either by pressing the ENTER key on the keyboard or clicking on the blue button with a paper plane icon next to the input field.

As this application is not intended to be a general purpose chatbot like Gemini or ChatGPT, the user should limit their messages to queries that are relevant to the data specification that they have chosen.

### Chatbot’s reply

If the user sends a message that does not relate to their chosen data specification, the chatbot will reply with a simple answer “I could not find any relevant information in the data specification for your message.”

If the chatbot can extract some relevant items from the data specification, its answer will contain the following:

* A SPARQL query (see Figure 6). The user can run this query against a graph database to retrieve data.
* A list of suggestions. These suggestions are the properties from the data specification which will expand the user’s current query (see Figure 7). Suggestions are grouped by items that they expand. For example, in Figure 7 the last three suggestions “accessibility for the elderly”, “braille” and “elevator in interior” expand the item “Barrier-free access”, which was mapped from the user message.

### Side panel

The right-side panel contains items that have been mapped from the user message. Each card in the panel is a class from the data specification that has been mapped. Each class contains the properties that have been mapped. For example, in Figure 8, the classes “Tourist destination” and “Barrier-free access” were mapped, and the property “bezbariérovost” of class Tourist destination was mapped. When more classes and properties are mapped, the UI will add them to this side panel.

### User message highlighting

When user sends a message, it gets added to the conversation history, as can be seen in Figure 5. When the chatbot sends an answer, the reply message is long, and the user message gets scrolled away. In this case, the UI displays the most recent user message directly above the text input field for user messages (see Figure 8). This message is displayed with certain words highlighted. These are the words that the chatbot has mapped to items from the data specification. The user can click on each highlighted word to display a summary.

Some items from the data specification may have been identified as playing a role in the user message but don’t necessarily have a direct mapping to the user message. These items are located under the “+1 referenced item” button below the message text in Figure 8. Upon clicking, the UI will display a list of items. Each item can be clicked on to see a summary for it.

### Suggested items

The user can add suggestions from the chatbot to their message, thus expanding it. The user selects suggestions by ticking the checkbox next to the suggestion they want to add. For each suggestion, the user can specify whether to add this item as OPTIONAL, which results in the item being in an OPTIONAL branch in the generated SPARQL query (see Figure 11).

When viewing a summary for a suggested item, the user can click on the “Add item to my message” button in the summary window (see Figure 10). This action is equivalent to tick the checkbox next to the suggested item in the chatbot’s reply.

#### Filtering datatype properties

If the user selects an item that is a datatype property (has simple data type), they can specify a filter expression in the text field below that selected item. The chatbot expects this to be a string, which it will insert into parentheses after the FILTER keyword in the generated SPARQL query. The user must use the string “{?var}” to refer to the property they are filtering. For example, if the filter expression given by the user is is **{?var} = “ABC”** like in Figure 11, the SPARQL query will have **FILTER (?braille = “ABC”)**.

#### Confirming selections

The user confirms all their selected items by clicking on the green “Add all selected items to my message”. This does not send a new message to the conversation. This action will prompt the chatbot to generate a suggested message, which is displayed to the user above the text input field. This suggested message is the user’s most recent message expanded by the items they have selected. If the user is happy with that message, they can send it to the chatbot without any modifications.

# Best practices

While the application does allow the user to modify the suggested message in the text input before sending it, this approach is generally not recommended. When the user sends their first message, the chatbot will have to retrieve the relevant items from the full data specification. When the user selects suggested items to add to their current message, the chatbot does not need to work with the full data specification but only the already mapped items and the suggestions. But if the user modifies the suggested message, the chatbot does not have a built-in mechanism to determine the extent of the change. The modified message could be completely different from the suggested message. In this case, the chatbot will have to work with the full data specification.

# Limitations

This chapter discusses the performance bottlenecks and unsupported scenarios in the application.

## Performance

Behind the scenes the chatbot works with a large language model (commonly referred to as AI) to map messages to the data specification items. Depending on the AI being used, the chatbot might take a long time to answer each user message.

## Limited query support

The system is unable to generate correct SPARQL queries for certain cases. For example, a query about a person who is a “boss of another person” would incorrectly translate into a query about a person who is their own boss.

# Glossary