Function Calling

Function Calling

You can register custom Java functions with the `OpenAiChatClient` and have the OpenAI model intelligently choose to output a JSON object containing arguments to call one or many of the registered functions. This is a powerful technique to connect the LLM capabilities with external tools and APIs. The models have been trained to detect when a function should to be called and to respond with JSON that adheres to the function signature.

Note that the OpenAI API does not call the function directly; instead, the model generates JSON that you can use to call the function in your code and return the result back to the model to complete the conversation.

To register your custom function you need to specify a function `name`, function `description` that helps the model to understand when to call the function, and the function call `signature` (as JSON schema) to let the model know what arguments the function expects. Then you can implement a function that takes the function call arguments from the model interacts with the external, 3rd party, services and returns the result back to the model.

Spring AI offers a generic ToolFunctionCallback.java interface and the companion AbstractToolFunctionCallback.java utility class to simplify the implementation and registration of Java callback functions.

Quick Start

Lets create a chatbot that answer questions by calling external tools. For example lets register a custom function that takes a location and returns the current weather in that location. Question such as "What's the weather like in Boston?" should trigger the model to call the function providing the location as an argument. The function uses some weather service API and returns the weather response back to the model to complete the conversation.

Let the MockWeatherService.java represent the 3-rd party weather service API:

JAVA
```
public class MockWeatherService implements Function<Request, Response> {

    public enum Unit { C, F }
    public record Request(String location, Unit unit) {}
    public record Response(double temp, Unit unit) {}

    public Response apply(Request request) {
        return new Response("30", Unit.C);
    }
}
```

Then extend AbstractToolFunctionCallback to implement our weather function like this:

JAVA
```
public class WeatherFunctionCallback
        extends AbstractToolFunctionCallback<Request, Response> {

    private final MockWeatherService weatherService = new MockWeatherService();

    public WeatherFunctionCallback(String name, String description, Class<Request> inputType) {
        super(name, // (1)
            description, // (2)
            inputType, // (3)
            (response) -> "" + response.temp() + response.unit()); // (4)
    }

    @Override
    public Response apply(Request request) {
        return this.weatherService.apply(request);
    }
};
```

The constructor takes a function name (1), description (2), input type signature (3) and a converter (4) to convert the `Response` into a text. The Spring AI auto-generates the JSON Scheme for the `MockWeatherService.Request.class` signature.

Registering Functions as Beans

If you enable the OpenAiChatClient Auto-Configuration, the easiest way to register a function is to created it as a bean in the Spring context:

JAVA
```
@Configuration
static class Config {
    @Bean
    public WeatherFunctionCallback weatherFunctionInfo() {
        return new WeatherFunctionCallback(
            "CurrentWeather", // (1) name
            "Get the weather in location", // (2) description
            MockWeatherService.Request.class); // (3) signature
    }
    ...
}
```

Now you can enable the `CurrentWeather` function in your prompt calls:

```java
OpenAiChatClient chatClient = ...

UserMessage userMessage = new UserMessage("What's the weather like in San Francisco, Tokyo, and Paris?");

ChatResponse response = chatClient.call(new Prompt(List.of(userMessage),
        OpenAiChatOptions.builder().withEnabledFunction("CurrentWeather").build())); // (1) Enable the function

logger.info("Response: {}", response);
```

Note

you must enable, explicitly, the functions to be used in the prompt request using the `OpenAiChatOptions.builder().withEnabledFunction(…)` method (1).

Above user question will trigger 3 calls to `CurrentWeather` function (one for each city) and the final response will be something like this:

```
Here is the current weather for the requested cities:
- San Francisco, CA: 30.0°C
- Tokyo, Japan: 10.0°C
- Paris, France: 15.0°C
```

The [ToolCallWithPromptFunctionRegistrationIT.java] integration test provides a complete example of how to register a function with the `OpenAiChatClient` using the auto-configuration.

Register/Call Functions with Prompt Options

In addition to the auto-configuration you can register callback functions, dynamically, with your Prompt requests:

```java
OpenAiChatClient chatClient = ...

UserMessage userMessage = new UserMessage("What's the weather like in San Francisco, Tokyo, and Paris?");

var promptOptions = OpenAiChatOptions.builder()
    .withToolCallbacks(List.of(new WeatherFunctionCallback(
            "CurrentWeather",
            "Get the weather in location",
            MockWeatherService.Request.class)))
    .build();

ChatResponse response = chatClient.call(new Prompt(List.of(userMessage), promptOptions));

logger.info("Response: {}", response);
```

Note

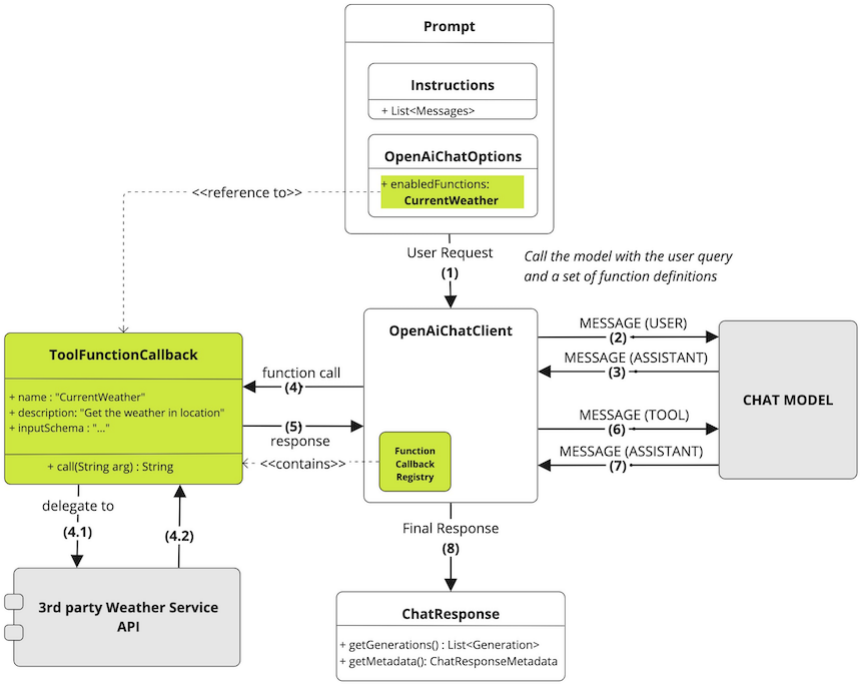
The in-prompt registered functions are enabled by default for the duration of this request.

This approach allows to dynamically chose different functions to be called based on the user input.

The [ToolCallWithPromptFunctionRegistrationIT.java] integration test provides a complete example of how to register a function with the `OpenAiChatClient` and use it in a prompt request.

Function Calling Flow

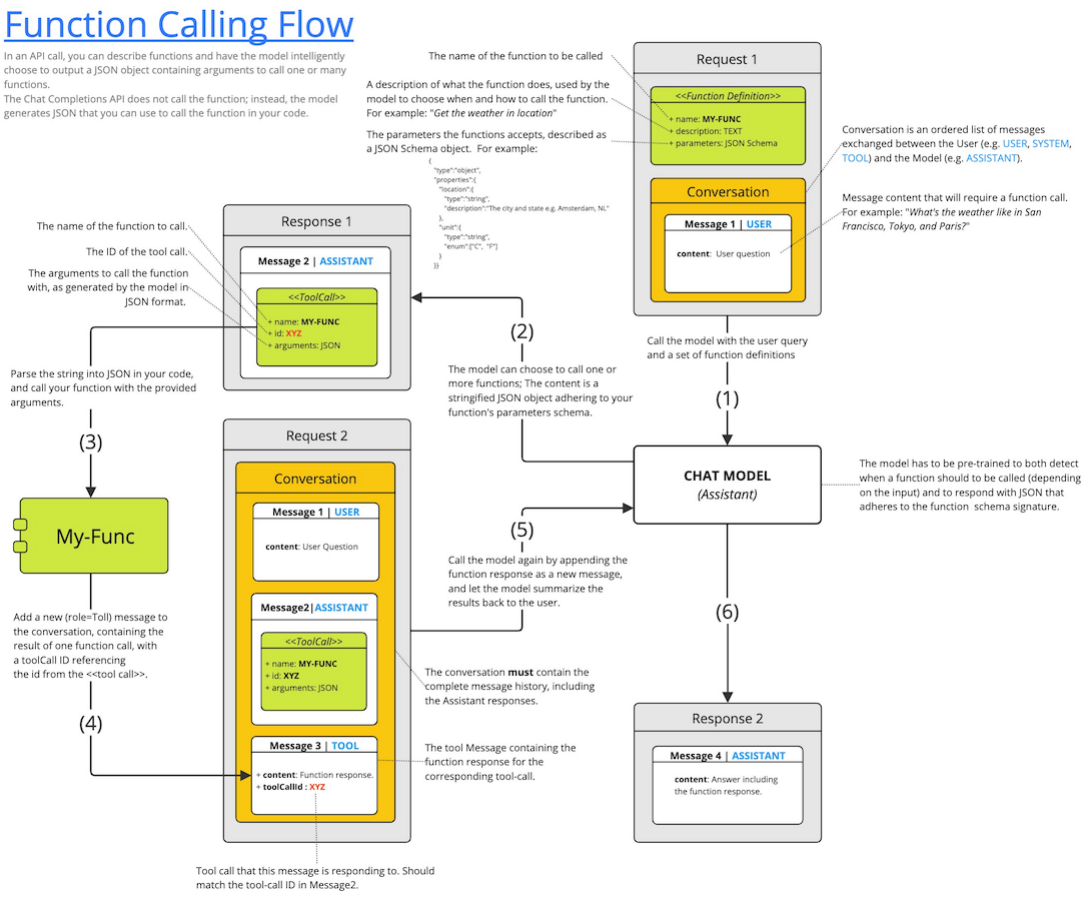
The following diagram illustrates the flow of the OpenAiChatClient Function Calling:



Appendices:

OpenAI API Function Calling Flow

The following diagram illustrates the flow of the OpenAI API Function Calling:



[org.springframework.ai.openai.chat.api.tool.OpenAiApiToolFunctionCallTests] provides a complete example of how to call a function using the OpenAI API. It is based on the OpenAI Function Calling tutorial.