

# Bot Gateway Server Quick Start Guide

How to develop a new bot for Bot Gateway Server

4/3/2026

# How to develop a new bot for Bot Gateway Server

## Prerequisites

1. Install Java Development Kit 1.8.
2. Install NetBeans 8.2 (https://netbeans.org/downloads/). You can use the edition listed as **Java SE** (alternatively, you can use any other Java IDE.)
3. Install "Apache Maven 3". You can either:
   - Install it from https://maven.apache.org/download.cgi.
   - Install a Java IDE with Maven embedded.
   - Reuse embedded Maven in NetBeans (in Windows the **mvn** application is located in the NetBeans installation folder under **java\maven\bin**).

## Preparing the environment for a new bot project

1. Obtain Bot Gateway Server (BGS) project template files (located in the subfolder **media-channel-drivers\channel-chatbot\provision** in the DMS installation folder).
2. Import the files into a local Maven repository by executing the following commands from the location where the project template files are stored:

```
> mvn org.apache.maven.plugins:maven-install-plugin:2.5.2:install-file
 -Dfile=ChatBotArchetype.jar -DpomFile=pom.xml

> mvn org.apache.maven.plugins:maven-install-plugin:2.5.2:install-file
 -Dfile=ChatBotApi.jar -Djavadoc=ChatBotApi-javadoc.jar
```

> **Important**
>
> You must either add the location of the **mvn** application into PATH or use the full-path specification when running the commands.

3. Update the archetype catalog in your local Maven repository with the following command:

```
> mvn archetype:update-local-catalog
```

> **Important**
>
> If the above command failed or if the archetype catalog was not created yet, run the following command:
>
> ```
> > mvn archetype:crawl
> ```

4. Restart NetBeans (if NetBeans is running).

## Creating a new bot project

1. Using NetBeans IDE, do the following:
   a. From the **File** menu select **New Project**.
   b. Select **Maven > Project from Archetype**.
   c. Click **Next**.
   d. Use the **Search** field to find the **ChatBotArchetype** archetype or select from the **Known Archetypes** list.
   e. Click **Next**.
   f. Modify the following:
      - **Project Name** - Name of the project (same as artifact ID)
      - **Project Location** - Where project will be placed
      - **Group Id** - Maven groupId
      - **Version** - Project version
      - **Package** (Optional) - Package for Java classes (should be generated based on **Group Id** and **Project Name**)
   g. Click **Finish**.
2. Implement your bot by following the directories from Bot implementation guidelines
3. Build the project:
   1. Right-click on the project from the **Projects** view.
   2. Select **Clean and Build**. The resulting JAR file is in the **target** subfolder of the project.

## Deploy a new bot

1. Copy the bot plugin JAR file to the subfolder **media-channel-drivers/channel-chatbot/bots-repo** in the DMS installation folder.
2. Enable the bot plugin. Open the DMS Application object and go to the section `[channel-chatbot-`

`monitor-bots]` (you must create the section if it is absent). Add an option where:

- The key contains the name of the bot plugin jar file (which is placed into the **bots-repo** folder). For example: `MySampleBot.jar`.
- The value provides the configuration for the bot, which is provided to the bot with method **initialize** of type **KeyValueMap**. This value can:
  - Contain `{}` (an empty configuration)
  - Contain a JSON string (enclosed between `{}` without line breaks)
  - Contain the full path to the file with JSON (line breaks allowed). The file name can be absolute or relative (in case of relative filename, the **bots-repo** directory is treated as a root folder). Use this approach if you need to provide sensitive data (for example, password) in bot configuration.

3. Observe the DMS logs to ensure the bot loads successfully.
4. Provide a workflow which starts, stops, and manages the bot.

> **Important**
>
> - If you want to disable the bot: Remove the corresponding option from the section `[channel-chatbot-monitor-bots]`.
> - If you need to update the configuration file for the bot:
>   - Preserve the original file by copying the JSON configuration file (a path to this file is specified in the DMS Appliction object) and renaming the copied file.
>   - Update the file as needed, then save your changes.
>   - Update the the corresponding option in section `[channel-chatbot-monitor-bots]` with the new filename.
> - If you need to replace the bot plugin JAR file, you must stop DMS, replace the JAR file, and then restart DMS.

# Bot implementation guidelines

## Bot implements two interfaces

- BGS invites bots into chat session via ChatBotFactory interface and then relays the chat session activity to the bot via the ChatBot interface. BGS also initializes and provides bot configuration through the ChatBotFactory interface. You must implement the bot's logic in the methods of the ChatBot interface.
  - You can modify and/or extend the default implementation of the ChatBotFactory interface. For example, you can change the `getBotId()` method if a different bot ID is needed.
  - When implementing methods of the ChatBot interface, you must use the ChatBotPlatform `cbpInstance` class in order to send messages or notices into the chat session, leave the chat

session, and update the userdata of interaction in workflow.

- If you need to change the name and/or package of the ChatBotFactory implementation class, update the file "com.genesyslab.chat.bots.chatbotapi.ChatBotFactory" in folder "src\main\resources\META-INF\services" of your project.
- The implementation of `ChatBot` and `ChatBotFactory` interface must not introduce new methods with the same names (and different parameters) which are already defined in the interface.
- It is recommended to use `ESP_REQUEST` as a source in `getUserData(UserDataSource)` of `getInteractionInfo()` in `GenesysChatSession` to avoid a possible race condition between workflow and bot launching.

- The bot must use the logger obtained through "com.genesyslab.chat.bots.chatbotapi.platform.LoggerFactory" or `ChatBotPlatform.getLogger()`. The bot must not use the "slf4j" logger directly.

> **Important**
>
> Since a single thread is being used to deliver all notifications to a bot, the methods implemented in the `ChatBot` and `ChatBotFactory` interfaces must not delay the thread which calls those methods, . Also, the esp-proc-timeout configuration option is what specifies the number of seconds to process all incoming ESP requests from the workflow.

## Bot uses interfaces provided by BGS

This functionality can be described as:

- Core functionality provided by ChatBotPlatform interface which allows a bot to send messages and notices to other participants, leave chat session, communicate to workflow (`updateUserdata`) and provide access to other interfaces. For guidance, you can check the implementation of basic bot functionality in EchoBot (the project files are located in the subfolder "media-channel-drivers\channel-chatbot\samples" of the DMS installation folder).
- File attachment API allowed to decipher notice events from other participants into file attachments (if notice represents it), to send a file attachment which can be created from raw file or from a file attached to a standard response. BGS IP contains `BalloonsBot` and `ThumbnailsBot` which demonstrate how to use the API for file attachments.
- Structured messages (also known as "rich" messages) API permits rich messages to be sent in a channel where those are supported and to process replies. BGS IP contains CafeBot which demonstrates the capabilities of this API, including the use of Standard Reponses for rich messages. Through the use of rich messages, the BGS API provides the `getRelatedEventId()` method which defines correspondence between sent structured messages and a received reply.

## High availability mode recommendations

By default, high availability mode does not require any special handling unless the bot needs to process special situations (these features are demonstrated in a new BobberBot sample project). In this case, the bot can:

- Use **HighAvailabilityManager** methods to save the bot context and to retrieve it upon bot restoration.

- Implement a **ChatDisconnectHandler** interface to process a disconnection and reconnection to a chat session.

**Notes:**

- BGS is a hosting environment for a bot JAR file which is simply a Java application. You can communicate with any 3rd party services (via REST HTTP API) from within the bot implementation, the same way as you can from any ordinary Java application.
- To implement agent escalations, you'll need to determine how the bot communicates with the workflow (in other words, URS/ORS strategies) which handles the corresponding interactions. For example, you can force the workflow to wait on a value condition of a certain userdata key/value pair (KVP), and then update this KVP from the bot when it needs to escalate to an agent. Or, alternatively, you can run the bot in so-called ”waiting” mode (see the sample workflow) and simply finish the bot execution (call `leaveSession()`) when it needs to escalate to an agent, so the workflow continues the routing.

## Using 3rd party libraries

- If your bot needs to use 3rd party libraries, you have the option to use `maven-assembly-plugin` and these libraries will be included into the resulting JAR file.
- Each bot is launched by a separately instantiated class loader to avoid issues such as unfulfilled JAR file dependencies or JAR version inconsistencies.