Genesys Intelligent Automation Bots Integration Guide

Genesys Intelligent Automation Current

11/9/2022

Table of Contents

Genesys Intelligent Automation Bot Integration Guide

This document provides information on integrating Genesys Intelligent Automation with chat bots and voice bots.

Important
The Natural Language Menu App and Custom Natural Language Menu are shipped separately and only allowed in conjunction with the purchase of either the bot bundle/orchestration either Voicebot or Chatbot bundle. Get in touch with your AE or Sales.

Genesys Intelligent Automation can provide your customers with chat or voice bot-based access by integrating with external artificial intelligence (AI) services to provide natural language understanding (NLU) capabilities.

When connected to a natural language engine, Intelligent Automation allows a user to use natural language when responding, even entering multiple pieces of information in a single response. The natural language engine infers the relevant information contained in a user's response, enabling some subsequent Menu and Question blocks in the call flow to be skipped, because the information has already been captured. The result is a successful conversation that is shorter than that of the directed-dialog approach.

Important
WebIVR does not support NLU.

Currently Intelligent Automation supports the following natural language services:

- Genesys Dialog Engine
- Google Dialogflow
- Microsoft Bot Framework with LUIS

Outline

- Prerequisites
- Bot Deployment
    - Voice
    - Chatbots

- Natural Language Systems
    - Dialog Engine
    - Dialogflow
    - Microsoft LUIS

Prerequisites for deploying bots

Chat and Voicebot Prerequisites

Genesys Engage Premise

Chatbots

- Genesys Intelligent Automation 9.0.100 or higher
- Genesys Mobile Services
- Chat Server
- DMS
- Bot Gateway
- Interaction Server
- Dialog Engine or third-party such as Google DialogFlow and Microsoft LUIS

VoiceBots

- Genesys Intelligent Automation 9.0.100 or higher
- Genesys Voice Platform 9.0.019.68 or higher
- Genesys Engage Routing
- SIP Server
- Resource Manager
- Genesys Dialog Engine or third party such as Google DialogFlow and Microsoft LUIS
- Google Speech-to-Text
- Google Text-to-Speech or Nuance Text-to-Speech

PureConnect Cloud & Premise

Chatbots

- Intelligent Automation v9.0.105 or higher
- PureConnect 2018R4 or higher
- Bot Messaging Service
- Genesys Widgets

- Dialog Engine or third party such as Google Dialogflow

VoiceBots

- Intelligent Automation v9.0.105 or higher
- PureConnect 2018R4 or higher. From PureConnect 2019R3 onwards, Intelligent Automation v9.0.106 or higher is required.
- CIC – integrated contact center software solution
- UniMRCP 1.5.0 or higher
- Google Speech Recognition plugin for UniMRCP
- Dialog Engine or third party such as Google Dialogflow
- Nuance and Google Text to Speech
- Google Speech-to-Text

Minimum Versions Required to Support Bot Applications

Bot Channel	Application	Minimum version	Notes
ALL	Genesys Intelligent Automation	9.0.104.00	PureConnect - 9.0.103.00
Voice	Genesys Media Control Platform	9.0.019.68	
Voice	UniMRCP (if required for TTS)	unimrcp-gss-1.6.1-1.el7.x86 PureConnect unimrcp-gsr-1.6.1-1el7.x86_64 (Speech to Text) unimrcp-gsr-1.6.1-1el7.x86_64 (Text to Speech)	Only required if prior to Google TTS support in MCP
Chat	Genesys Widgets	9.0.009.04	Assumption: Rich Messaging required on Chat.
Chat	Genesys Mobile Services	8.5.201.04	Assumption: Rich Messaging required on Chat.
Messaging	Genesys Digital Messaging Server	9.0.002.06	
Chat / Messaging	Genesys Chat Server	8.5.109.06	
Chat / Messaging	Genesys Chat Server	8.5.203.09	If GCXI Bot Reporting is required
Chat / Messaging	Genesys Bot Gateway	9.0.004.08	Refer to the latest

Bot Channel	Application	Minimum version	Notes
	Server		version here.
Chat / Messaging	Genesys Digital Messaging Server	9.0.001.10	For BGS driver
Chat / Messaging	Genesys eServices Manager	8.5.303+	If Rich Messaging SRLs in UCS is required.
Chat / Messaging	Genesys UCS	8.5.200.19	If Rich Messaging SRLs in UCS is required.

Deploying bots in Intelligent Automation

Before you can deploy a bot in Intelligent Automation, ensure that you have the required prerequisites.

Deploying Chat Bots

Genesys Intelligent Automation can provide your customers with chat bot-based access to your WebIVR applications. Customers can converse with a bot using natural language to accomplish various business needs, such as making a payment or checking an account balance.

Warning

Please note that the customer is responsible for ensuring that the environment and bot applications they build are properly configured and secured according to PII and HIPAA requirements.

PureConnect

For information on deploying PureConnect chatbots, see PureConnect with Genesys Intelligent Automation for more information.

Genesys Engage

The following components are required:

- Digital Messaging Server 9.0.x - Provides the bot platform.
- Genesys Administrator - Used for configuring the chat bot client.
- Interaction Routing Designer - Provides the routing strategy.

Important

For SMS (Short Message Service) functionality, you must also install Social Messaging Server 8.5.x with SMS plugin.

Important

To test your bot functionality using Virtual Call, please set the **Channel** field to `WEBCHAT` when starting your Virtual Call session..

Once installed, the bot acts similarly to a WebIVR application using prompts from the associated visual persona. However, if using SMS, remember that customers cannot use special characters in response to queries from the bot.

Preparing the chatbot environment

Install Digital Messaging Server

1. Refer to the Digital Messaging Server Guide for information on how to install Digital Messaging Server.
2. Copy the chatbot jar file (fish-cbp-<version>.jar) to the Digital Messaging Server subfolder for chatbots. For example: **C:\Program Files\GCTI\eServices 9.0\Digital Messaging Server\DMS\media-channel-drivers\channel-chatbot\bots-repo\**.

Use Genesys Administrator to configure Digital Messaging Server

1. Open Genesys Administrator and log in to your configuration environment.
2. Go to **Provisioning > Environment > Applications** and open the Digital Messaging Server object.
3. Create a section called `channel-chatbot-monitor-bots`.
4. Create a chatbot option. You must ensure the option name exactly matches the name of the jar file that you previously uploaded to Digital Messaging Server. For example: `fish-cbp-9.0.0.jar`.
5. Set the value of the chatbot option to point to the load balancer for VUI Server. You can also add the parameters **downstream_request_timeout_millis** and **session_timeout_minutes** to set timeout values. In the following example, the request timeout is **120000** milliseconds (120 seconds) and the session timeout is **30** minutes:

```
{
 "base_url" : "http://www.yourcompany.com:8081/fish-vui",
 "downstream_request_timeout_millis" : 120000,
 "session_timeout_minutes" : 30
 }
```

Important

Set the **RPC cache timeout** field value to zero. See RPC cache timeout for detailed explanation on this setting.

6. Click **Save & Close**.

Important

Starting from 9.0.109.00 release, IA provides support for proxy servers. The following optional parameters can be passed to the **fish-cbp** jar file:

- **x_vui_proxy_host** - The proxy server URL or IP address
- **x_vui_proxy_port** - The port number
- **x_vui_proxy_username** and **x_vui_proxy_password** - The credentials to access the proxy server. Required only when the proxy server requires authentication.

Example:

```
{
 "base_url" : "http://www.yourcompany.com:8081/fish-vui",
 "downstream_request_timeout_millis" : 120000,
 "session_timeout_minutes" : 30,
 "x_vui_proxy_host" : "proxy_server_url",
 "x_vui_proxy_port" : "port_number",
 "x_vui_proxy_username": "username",
 "x_vui_proxy_password": "password"
 }
```

If the proxy server information is not configured or are empty, the VUI server will be accessed directly.

Starting from 9.0.109.02 release, IA supports configuring the default error message that is displayed when the connection between the Chat Bot Platform (CBP) and VUI fails. The new optional JSON entity, **x_vui_default_error_msg**, can be used to configure the message.

Starting from 9.0.110.xx release, IA supports configuring the connection pool mechanism with three new optional parameters: `max_total_http_connections`, `max_route_http_connections`, and `connection_pool_maintenance_delay_millis`.

Parameter Name	Description	Default Value
max_total_http_connections	The maximum total number of idle and borrowed connections that can be active at the same time.	200
max_route_http_connections	The maximum limit of connections on a per-route (host) basis.	200
connection_pool_maintenance_delay_millis	The time delay in milli-seconds after which a pool's connection is validated and expired connections are terminated.	500

```
{
...
"max_total_http_connections" : 200,
"max_route_http_connectionsi" : 100,
"connection_pool_maintenance_delay_millis" : 1000,
...
}
```

(Optional) Use Genesys Administrator to configure SMS Server

Important

- Use this section only if you want to provide SMS (Short Message Service) access to your chatbot. Otherwise, go to the next section.
- Refer to SMS Server (part of eServices) documentation for more information on SMS Server.
- This feature uses the *Session Mode* of SMS Server, which refers to creating and keeping an interactive conversation between a mobile client and an agent in the form of a conventional chat session. All messages received and sent during this session are associated with one interaction, which corresponds to this SMS session.

1. Open Genesys Administrator and log in to your configuration environment.
2. Go to **Provisioning > Environment > Applications** and open the SMS Server object.
3. Open the channel that you want to use with the bot.
4. For the option **inbound-route**, specify the access point in your routing strategy that is used to place submitted interactions for incoming messages.
5. Click **Save & Close**.

Use Interaction Routing Designer to configure the chatbot

Once you have installed the bot interface, you can use Interaction Routing Designer to configure settings.

1. Open Interaction Routing Designer and log in to your environment.
2. Open the routing strategy that is triggered by the *session* endpoint, and add an **External service** block to the start of the flow.
3. Open the **External service** block and go to the **General** tab. Configure the settings as described below:
   - **Application type** - Set to the Digital Messaging Server or Social Messaging Server in your environment.
   - **Application name** - Set to the Digital Messaging Server or Social Messaging Server in your environment.
   - **Service** - Set to `ChatBotPlatform`.
   - **Method** - Set to `StartBot`.

In the **Parameters** section, add the following:

- **Nickname** - The bot name displayed to customers.
- **SiteID** - The ID number of your Genesys Intelligent Automation application to use with the bot.
- **IsTestCall** - If **true**, the test version of the application is used. Otherwise, set to **false** to use the production version.
- **AuthToken** - The value of the **Authentication Key** field in your company page.
- **StopBotOnAgentArrival** - Set to **true** if you want the bot to end once an agent joins the session. Otherwise, set to **false**.
- **StopBotOnCustomerLeft** - Set to **true** if you want the bot session to end once a customer leaves the session. Otherwise, set to **false**.
- **Visibility** - Set to **ALL**.
- **ChatBotID** - Set to `speechstorm-chatbot`.
- **ChatBotName** - Do not enter information in this field.
- **_umsChannel** - Set to `channel-chatbot`.

4. Click **OK**.

Genesys recommends that you add new blocks to your routing strategy, just after the **External service** block, to pause the strategy until the chatbot has completed its task. Otherwise, the routing strategy might queue up the session for an agent and, as soon as an agent joins, the **StopBotOnAgentArrival** parameter will cause the bot to terminate early.

To determine if a chatbot interaction is still alive, Genesys Intelligent Automation checks the following interaction data values:

- **IsOnline** - If the value is **0**, the bot is offline and you can terminate the interaction.
- **FishTransferRequested** - If the value is **true**, you can proceed to route the interaction to an agent.

Refer to the Interaction Routing Designer documentation or consult your Genesys representative for information on the best routing strategy for your environment.

Deploying Voice Bots

VoiceBots (previously known as Cognitive IVR) uses Google Cloud Speech-to-Text to improve the performance of natural-language interfaces such as Dialog Engine. This makes it easier for callers to use spoken natural-language phrases to navigate through an Genesys Intelligent Automation application.

Warning

Please note that the customer is responsible for ensuring that the environment and bot applications they build are properly configured and secured according to PII and HIPAA requirements.

For example, a traditional IVR might have trouble interpreting the following phrase: "I would like to book a flight to Paris on Friday afternoon." Instead, the traditional IVR would need to ask several sequential questions to deduce the following information:

- What do you want to do? (Book a flight)
- Where do you want to travel? (Paris)
- When do you want to travel? (Friday afternoon)

However, VoiceBots can interpret and parse natural language so the customer can make the same one-sentence request without having to sequentially answer several questions: "I would like to **book a flight** to **Paris** on **Friday afternoon**."

Other than a new login screen, using VoiceBots does not affect how you use Genesys Intelligent Automation. You can build and use applications in the same way (but note the limitations below) and share the same database.

Integrate Intelligent Automation with Google Speech-to-Text

When setting up Intelligent Automation for use with voice, a start page is required.

Speech-to-Text Engine	Text-to-Speech Engine	Product	Version	Start Page
Google Cloud for Speech Recognition	Nuance **Important** Currently IA supports Nuance version 7.2 on GVP	Genesys Voice Platform	8 and 9	GenesysGVP8_GoogleSR_Vocalizer
		PureConnect	2018R4 - 2019R2	Interactive_GoogleSR_Vocalizer5.j
		PureConnect	2019R3 and above	Interactive_GoogleSR_Vocalizer5_J

Speech-to-Text Engine	Text-to-Speech Engine	Product	Version	Start Page
	only.			
	Google	Genesys Voice Platform	8 and 9	GenesysGVP8_GoogleSR_GoogleS
		PureConnect	2018R4 - 2019R2	Interactive_GoogleSR_GoogleSS.js
		PureConnect	2019R3 and above	Interactive_GoogleSR_GoogleSS_J

The start page should contain the following parameters:

- `testsiteid` - The ID of the application being provisioned.
- `istestcall` - When set to *True*, the provisioning uses the test mode (the latest saved version) or the production mode (the latest deployed version) when set to *False*.
- `authtoken` - This will be available from the **Company Details** page.
- `mrcpversion` - This is an optional parameter. The default value is *1*.

An example start page that uses GVP: *http://<server>:<port>/fish-vui/start/ GenesysGVP8_GoogleSR_Vocalizer6.jsp?testsiteid=53&istestcall=true&authtoken=303a935e028b1aae234476fed4*

The RTP is streamed to Google STT and the resulting transcription is sent to Dialog Engine or DialogFlow for processing.

Support Phrase Hints

Important

- Phrase hints are not supported in PureConnect.
- Phrase hints are limited to 100 characters in Genesys Engage.
- To enable Phrase Hints, create a new option, **FeatureEnablement.PhraseHints** and set it to *True* in **Server Settings**
- Verify that the **FeatureEnablement.General.HiddenFields** option does not include any references to Phrase Hints. If present, remove the entry and restart the GUI to enable the Phrase Hints feature.
- Phrase hints are available in the languages defined in the server setting **VuiPreferences.AvailableLanguages.ASR** and will be in the same languages used for personas.
- Phrase Hints is not supported for Menu blocks. They are only supported for Question blocks using the *Any Text* grammar.

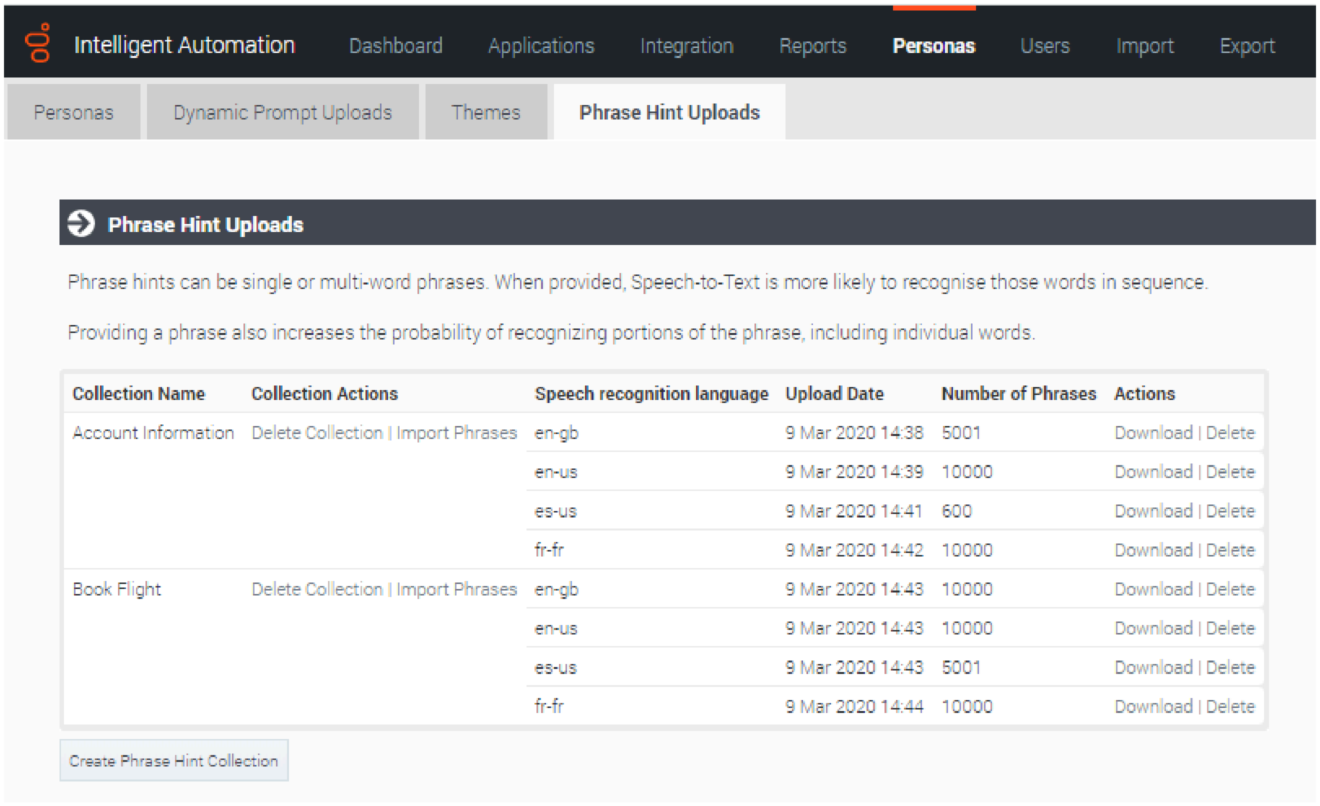
Intelligent Automation supports specifying domain-specific phrase hints as a way to provide context and to improve speech recognition. You can create a collection of phrases that can contain multiple lists of phrases for each language.

You can upload a list of phrases as a Comma-Separated Values (CSV) file from the **Phrase Hints Upload** page. You can update the list or collection or remove the phrase lists or collections.

These phrase hints are available for Question blocks only in callflows. To use this feature, the **Standard Grammar** option must be set to *Any Text*. Additionally, you can set the collection to be used in the **Preferences** screen.

Important

If you have several persons configured, ensure that the **Standard Grammar** option must be set to *Any Text* for all personas. If one persona has a different grammar configured, Phrase Hints are ignored.



Phrase hints can be single or multi-word phrases. When provided, Speech-to-Text is more likely to recognise those words in sequence.

Providing a phrase also increases the probability of recognizing portions of the phrase, including individual words.

Collection Name	Collection Actions	Speech recognition language	Upload Date	Number of Phrases	Actions
Account Information	Delete Collection | Import Phrases	en-gb	9 Mar 2020 14:38	5001	Download | Delete
		en-us	9 Mar 2020 14:39	10000	Download | Delete
		es-us	9 Mar 2020 14:41	600	Download | Delete
		fr-fr	9 Mar 2020 14:42	10000	Download | Delete
Book Flight	Delete Collection | Import Phrases	en-gb	9 Mar 2020 14:43	10000	Download | Delete
		en-us	9 Mar 2020 14:43	10000	Download | Delete
		es-us	9 Mar 2020 14:43	5001	Download | Delete
		fr-fr	9 Mar 2020 14:44	10000	Download | Delete

The following video demonstrates how to use Phrase Hints (from 02:03 onwards).

Link to video

Limitations

The following limitations apply to the use of VoiceBots:

- Multimodal communication is not supported.
- The **Release ASR** API command for the Script block does not have any effect when using VoiceBots.
- WebIVR applications are not supported.
- For a given persona, the TTS language is also used for the speech-recognition language.

Configuring Voicebots

Using UniMRCP with PureConnect

This section describes how to connect to the PureConnect platform and enabling the enhanced and phone call models in UniMRCP/

- In Intelligent Automation, set the VUI Preference **Collection low confidence threshold** to `0`.
- Under the Defaults section of the main application:
  1. Set the recognition incomplete timeout to 50000ms (this is only required when using UniMRCP).
  2. Set the **Collection Low Confidence** threshold to `0.0` in the IA application defaults.
  3. Turn on the **enhanced model** in UniMRCP config.
  4. Turn on the **phone_call model** in UniMRCP config.
- Ensure both the enhanced and phone_call model configs are enabled within UniMRCP.

The Phrase Hints feature (introduced in 9.0.108.00) and the Speech Contexts feature in UniMRCP can help in enabling better contextual understanding like differentiating between *Agent* and *Asian*.

Tip

Understanding accents that are not in the language set as default can affect performance and must be considered when developing voicebots.

Specifying a voice in PureConnect Cloud

Starting from release 9.106, you can specify a voice in GIA without specifying it in UniMRCP, you need to specify the desired voices in Interaction Administrator. See Phrase hints for more information.

UniMCRP Notes

How to control Pitch and Speed control

There is a UniMRCP patch for gss 1.7.3 – you can now globally configure the prosody rate and pitch at the UniMRCP config file.

Changing voices

Important

You can change voices only when Genesys Voice Platform MCP is configured with Google Speech resources.

Configure the **Default Server Settings** for your locale e.g., `Personas.Google.de-de.TTSVoiceName = e-DE-Standard-A,de-DE-Standard-B,de-DE-Wavenet-A,de-DE-Wavenet-B,de-DE-Wavenet-C,de-DE-Wavenet-D`. After this configuration, apply the setting in the Persona configuration.

Support for gRPC

GVP supports a gRPC integration with Google Cloud Text-to-Speech and Speech-to-Text for Voicebots purchased from Genesys.

Important

Contact Genesys Product Management for information on configuring GVP to connect to Genesys Voicebots.

Improving Confidence Scores

If the confidence score returned from GVP is below the threshold set in Intelligent Automation, reduce the threshold value to 0. This is applicable when GVP returns a *no match* value.

Using Intelligent Automation with NLUs

Intelligent Automation can integrate with external artificial intelligence (AI) services to provide natural language understanding (NLU) capabilities. Intelligent Automation currently has built-in connections to:

- Genesys Dialog Engine
- Google Dialogflow
- Microsoft Bot Framework with LUIS

When Intelligent Automation connects to a natural language engine, Intelligent Automation allows a user to use natural language when responding, even entering multiple pieces of information in a single response. The natural language engine infers the relevant information contained in a user's response, enabling some subsequent Menu and Question blocks in the call flow to be skipped, because the information has already been captured. The result is a successful conversation that is shorter than that of the directed-dialog approach.

Warning

Please note that the customer is responsible for ensuring that the environment and bot applications they build are properly configured and secured according to PII and HIPAA requirements.

How it works

The natural language engine works in the background while Intelligent Automation communicates directly with the user. Here's how:

1. Intelligent Automation asks an open-ended question (For example, *What can I help you with?*)
2. The user provides a natural language response (For example, *What will the weather be like in London on Saturday*?).
3. Intelligent Automation receives the response and passes it along to the natural language engine.
4. The natural language engine responds, as follows:
   - If the natural language engine needs more information to understand the request, it responds to Intelligent Automation with follow-up questions. (For example, *For what city?*).
   - If the natural language engine has all the information it needs, it sends Intelligent Automation the Intent and associated Slots (In the Weather example, the Intent could be Weather and the Slots could be weatherLocation and weatherDate).
5. When Intelligent Automation receives and reads the Intents and Slots, it directs the interaction according to the configured application.

The following demonstrates a chat session with a natural language engine running in the background:

Link to video

Supported NLU Features

Supported Feature	Genesys Dialog Engine	Google Dialogflow	Microsoft Bot Framework
Map Intents to Modules	✓	✓	✓
Map Slots to questions	✓	✓	X
Support for Live Agent Hand-off	X	✓	X
Support for Maximum Attempts	✓	✓	X
Pass Context Settings	✓	✓	✓
Bypass Opening Question	✓	✓	✓
Support for System Entities	X	✓	X
Support for Follow-up Intents	X	✓	X

What next?

To configure Intelligent Automation, read the following topics:

- Integration with Genesys Dialog Engine
- Integration with Google Dialogflow
- Integration with Microsoft Bot Framework and LUIS

Important
If you want to use other natural language AI services, contact your Genesys representative for help with a custom integration.

Integrating Intelligent Automation with Dialog Engine

Important

This integration is available only for existing Dialog Engine customers. Contact your account manager if you want to integrate Intelligent Automation with Dialog Engine.

Configuring IA with GDE

To integrate with Genesys Dialog Engine, complete the following steps:

1. Configure Default Server Settings
2. Configure NLU Settings
3. Map Intents to modules
4. Map Slots to questions

Configure Default Server Settings

If you're using Genesys Dialog Engine, configure the following server settings:

- **DialogEngine.JOPv3.AuthBaseURL** - The URL to the Genesys Cloud Authentication API.
- **DialogEngine.JOPv3.FetchTimeoutMillis** - The length of time (in milliseconds) that Intelligent Automation waits for a response from Dialog Engine before throwing an error.
- **DialogEngine.JOPv3.ClientID** - The Dialog Engine Client Secret password.
- **DialogEngine.JOPv3.ClientSecret** - The Dialog Engine Client Secret password.

Important

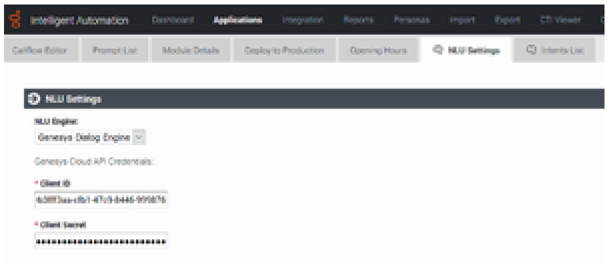
If you do not have access to these settings, contact your Genesys representative.

Configure NLU Settings

Once you have configured the Default Server Settings, open a **Natural Language Menu** module and click the **NLU Settings** tab.

From the **NLU Engine** menu, select Genesys Dialog Engine.

- Leave the **Use default credentials** box checked to use these credentials or uncheck the box to override them for this particular application.



NLU Settings

NLU Engine: Genesys Dialog Engine

Genesys Cloud API Credentials:

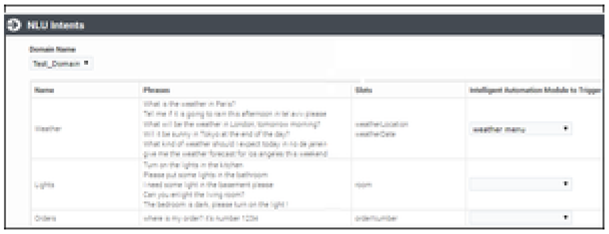
* Client ID: 4c3fff3aa-cfb1-47c9-8446-999876

* Client Secret: ••••••••••••••••••••••••

Map Intents to modules

Intelligent Automation reads all Intents, Utterances, and Slots associated with a domain and then displays that information on the **Intents List** page for the **Natural Language Menu** module. This is where you'll map each Intent to a module.

To map an Intent to a module, simply select a module from the **Intelligent Automation Module to Trigger** menu for the Intent. Intelligent Automation will then automatically update the application's callflow accordingly.



NLU Intents

Domain Name: Test_Domain

Name	Phrases	Slots	Intelligent Automation Module to Trigger
Weather	What is the weather in Paris? Tell me if it is going to rain this afternoon in tel aviv please What will be the weather in London, tomorrow morning? Will it be sunny in Tokyo at the end of the day? What kind of weather should I expect today in rio de janei give me the weather forecast for los angeles this weekend	weatherLocation weatherDate	weather menu
Lights	Turn on the lights in the kitchen Please put some lights in the bathroom I need some light in the basement please Can you enlight the living room? The bedroom is dark, please turn on the light !	room	
Orders	where is my order? its number 1234	orderNumber	

> **Important**
>
> If the fulfillment is being performed by the bot, you can select *None* from the **Intelligent Automation Module to Trigger** menu and a generic “success” result will be returned after the intent has been completed.

Map Slots to questions

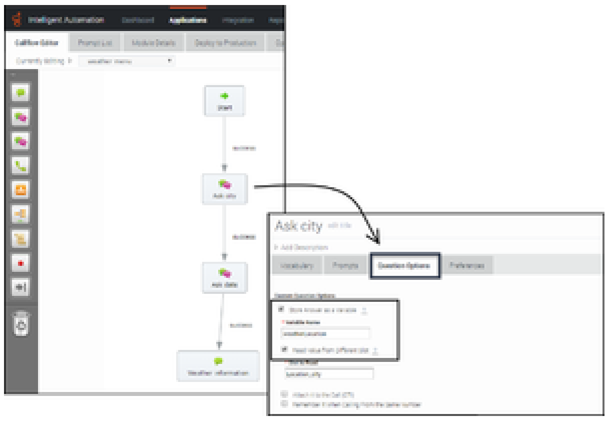
In the natural language engine, an Intent contains Slots, which are key pieces of information you need to extract from the end user to process a request. For example, if a user wants the weather forecast, you would need to know two key pieces of information before providing a weather forecast - city and date. These would be considered Slots.

For each Intent, you should have an associated module containing questions that will extract the right information from the customer. For the Weather example, you would have a Weather Questions module, which contains two questions: *For what city* and *For what date*? Both of these questions relate to the weatherLocation and weatherDate Slots.

Once you have created this module in Intelligent Automation, you need to map it to its associated Slots, as follows:

1. Open the module that is linked to Intent.
2. For each Question block in that module for which you would expect to have a slot, open the block and go to **Question Options**.
3. Check the **Store Answer as a Variable** checkbox and enter the Slot name (weatherLocation).

When a chat or voice session reaches one of those Question blocks, it first checks if that slot has been sent to Intelligent Automation from the natural language engine. In that case, Intelligent Automation uses that as the answer to the question. Otherwise, Intelligent Automation asks the question and waits for a response.



Supported NLU Features

Currently, Intelligent Automation supports the following Dialog Engine features:

- Support for Maximum Retries
- Pass Context Settings
- Bypass Opening Question

Support for Maximum Attempts

You can set the maximum number of retries to fill a slot and the maximum number of attempts to identify intents in a chat session.

The **Enable Maximum Attempts Counters** setting allows configuring the number of retry attempts and the action to be performed when the limit is reached.

The **Enforce Maximum Attempts for filling Mandatory Slot** will enable or disable the slot fill failure timeout feature. The number of unsuccessful tries can be configured after which the call is either transferred to a different module or return a result.

For example, if the number of retries is specified as 2, the bot will look if it can fill up any slots based on the user input two times. If the slot is not filled after both attempts, the call flow will be transferred to either a different module or return a special value.

When a value is provided in the **Maximum Attempts to Fill Mandatory Slot Values** option, the bot will try to slot-fill the user input. If the bot cannot perform a slot-fill successfully, the counter is increased. When the counter value exceeds the maximum retry values, Intelligent Automation either transfers the callflow to a different module (configured in the **Exit Module** field) or return a special result.

> **Important**
> When a slot is filled successfully or if the intent is switched, the counter is reset to zero.

When the **Enforce Maximum Attempts for Intent Disambiguation** option is configured, the bot will try to identify an intent from the conversation. When the maximum number of unsuccessful attempts is reached, Intelligent Automation either transfers the callflow to a different module (configured in the **Exit Module** field) or return a special result.

Pass Context Settings

Intelligent Automation allows you to pass variables to an NLU engine. The NLU engine can use the variables as part of the slot filling capabilities when the **Send Intelligent Automation variables as context to NLU Engine** setting is enabled.

For Dialog Engine, all variables are passed and Dialog Engine will map the variables to slots if the variable names from Intelligent Automation match the slot names in Dialog Engine.

Bypass Opening Question

You can allow users to bypass the initial question from the bot by passing the utterance as a variable to the NLU. The NLU then skips the initial question and proceeds to the next question. You can use any previously collected data or create and define a variable in the Script block and pass the utterance as a parameter in the **Link** block to the Natural Language Menu. The utterance is then passed to the NLU menu as an input in the callflow.

Integrating Intelligent Automation with Google Dialogflow

Configuring IA with Dialogflow

To integrate with Google Dialogflow, complete the following steps:

1. Configure NLU Settings
2. Map Intents to modules
3. Map Slots to questions

Important

- Before configuring the bots, please ensure that the server time and the local server time are synchronised.
- Proxy servers to access *.googleapis.com are not supported.
- Only HTTPS protocols are supported. For Natural Language Modules, Java clients are supported. For Custom Natural Language Module, REST clients are supported.
- By default, the 443 port is used by Google Services.

Configure NLU Settings

Open a **Natural Language Menu** module and click the **NLU Settings** tab.

From the **NLU Engine** menu, select Google Dialogflow.

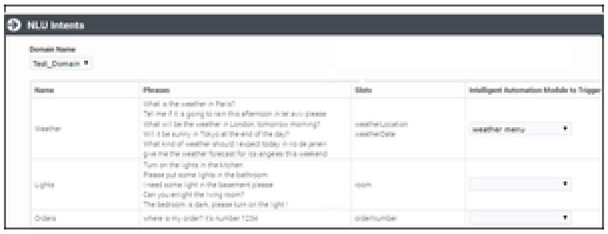
- Leave the **Use default credentials** option checked to use these credentials or clear the option to override them for this particular application.

Map Intents to modules

Intelligent Automation reads all Intents, Utterances, and Slots associated with a domain and then displays that information on the **Intents List** page for the **Natural Language Menu** module. This is where you'll map each Intent to a module.

To map an Intent to a module, simply select a module from the **Intelligent Automation Module to**

Trigger menu for the Intent. Intelligent Automation will then automatically update the application's callflow accordingly.



NLU Intents

Domain Name

Test_Domain

Name	Phrases	Slots	Intelligent Automation Module to Trigger
Weather	What is the weather in Paris? Tell me if it is going to rain this afternoon in tel aviv please What will be the weather in London, tomorrow morning? Will it be sunny in "Tokyo at the end of the day?" What kind of weather should I expect today in rio de janeiro give me the weather forecast for los angeles this weekend	weatherLocation weatherDate	weather menu
Lights	Turn on the lights in the kitchen Please put some lights in the bathroom I need some light in the basement please Can you enlight the living room? The bedroom is dark, please turn on the light !	room	
Orders	where is my order? its number 1234	orderNumber	

Important

If the fulfillment is being performed by the bot, you can select *None* from the **Intelligent Automation Module to Trigger** menu and a generic “success” result will be returned after the intent has been completed.

Map Slots to questions

In the natural language engine, an Intent contains Slots, which are key pieces of information you need to extract from the end user to process a request. For example, if a user wants the weather forecast, you would need to know two key pieces of information before providing a weather forecast - city and date. These would be considered Slots.

For each Intent, you should have an associated module containing questions that will extract the right information from the customer. For the Weather example, you would have a Weather Questions module, which contains two questions: *For what city* and *For what date*? Both of these questions relate to the weatherLocation and weatherDate Slots.

Once you have created this module in Intelligent Automation, you need to map it to its associated Slots, as follows:

1. Open the module that is linked to Intent.
2. For each Question block in that module for which you would expect to have a slot, open the block and go to **Question Options**.
3. Check the **Store Answer as a Variable** checkbox and enter the Slot name (weatherLocation).

When a chat or voice session reaches one of those Question blocks, it first checks if that slot has been sent to Intelligent Automation from the natural language engine. In that case, Intelligent Automation uses that as the answer to the question. Otherwise, Intelligent Automation asks the question and waits for a response.

Supported NLU Features

Currently, Intelligent Automation supports the following Dialogflow features:

- Support for Live Agent Hand-off
- Support for Maximum Retries
- Pass Context Settings
- Bypass Opening Question
- Support Follow-up Intents
- Support Fallback Intents
- Support Phrase Hints
- Support for System entities

Support for Live Agent Hand-off

When the live agent hand-off flag is returned as part of the DialogFlow intent, you can configure IA to follow a special path, usually a hand-off to a live agent.

When a value is provided in the **Path to take When Agent Requested** setting, the call is transferred to a live agent defined in the callflow.

In Dialogflow, the liveAgentHandoff flag should be enabled using a Custom Payload response within the intent.

```
{ “liveAgentHandoff” : true }
```

Support for Maximum Attempts

You can set the maximum number of retries to fill a slot and the maximum number of attempts to identify intents in a chat session.

The **Enable Maximum Attempts Counters** setting allows configuring the number of retry attempts and the action to be performed when the limit is reached.

The **Enforce Maximum Attempts for filling Mandatory Slot** will enable or disable the slot fill failure timeout feature. The number of unsuccessful tries can be configured after which the call is either transferred to a different module or return a result.

For example, if the number of retries is specified as 2, the bot will look if it can fill up any slots based on the user input two times. If the slot is not filled after both attempts, the call flow will be transferred to either a different module or return a special value.

When a value is provided in the **Maximum Attempts to Fill Mandatory Slot Values** option, the bot will try to slot-fill the user input. If the bot cannot perform a slot-fill successfully, the counter is increased. When the counter value exceeds the maximum retry values, Intelligent Automation either transfers the callflow to a different module (configured in the **Exit Module** field) or return a special result.

Important

When a slot is filled successfully or if the intent is switched, the counter is reset to zero.

When the **Enforce Maximum Attempts for Intent Disambiguation** option is configured, the bot will try to identify an intent from the conversation. When the maximum number of unsuccessful attempts is reached, Intelligent Automation either transfers the callflow to a different module (configured in the **Exit Module** field) or return a special result.

Pass Context Settings

Intelligent Automation allows you to pass variables to an NLU engine. The NLU engine can use the variables as part of the slot filling capabilities when the **Send Intelligent Automation variables as context to NLU Engine** setting is enabled.

For DialogFlow, the parameter's default value must be mapped to the variable defined in Intelligent Automation. You can configure the Default Value option from the contextual menu for a parameter in DialogFlow and it should be use the following format: *#fish_context.<VariableName>*.

Bypass Opening Question

You can allow users to bypass the initial question from the bot by passing the utterance as a variable to the NLU. The NLU then skips the initial question and proceeds to the next question. You can use any previously collected data or create and define a variable in the Script block and pass the utterance as a parameter in the **Link** block to the Natural Language Menu. The utterance is then passed to the NLU menu as an input in the callflow.

Support for Follow-up Intents

Intelligent Automation now supports follow-ups intents from Google Dialogflow. A follow-up intent is an intent that is tied to another earlier intent. Think of it as a child intent of an existing parent intent. When you create a follow-up intents in Dialogflow, Intelligent Automation can read and work with contexts for the intents.

Set up a module with a **Question** block. The Question block stores the intent in the **Prompt Wording** field. Ensure that this field has `var:NLTextResponse` as its value. In the **Question Options** tab, enable the **Store Answer as a variable** option. This holds the values received from DialogFlow. Ensure that the variable name is `utterance`.

Support for Fallback Intents

If a fallback intent is detected, IA will refer to the **Ask Natural Language Question** and return the fallback text that was configured in Dialogflow. This text could be something like: *Sorry, I wasn't able to understand that. Can you try explaining again?*. Users can make another attempt to provide information.

The **Enable Maximum Attempts Counters** and **Enforce Maximum Attempts for filling Mandatory Slot** options allow counting the number of attempts and either return a result or redirect to a different module.

Support Phrase Hints

See Phrase Hints for more information.

Support for System entities

Important

This feature is supported only for Google Dialogflow.

DialogFlow supports composite entities and returns them as part of the response. To allow use of these entities in a callflow, Intelligent Automation can use the values as a whole object or split the object into simpler components.

For example, in DialogFlow, we have a slot called *Duration* and the data type is `@sys.duration`, DialogFlow returns the following object as a result:

```
{"amount":10,"unit":"min"}
```

Intelligent Automation will store the object as a slot which can be accessed using a **Script** block.

```
Duration  = {"amount":10,"unit":"min"}
```

Intelligent Automation will also split the object as separate sub-slots which can be referenced in a **Question** block (see section Map Slots to questions). You can access each entity as `<slotName_type> = value`.

```
Duration_amount = 10
Duration_unit = min
```

Sample script

This script will parse the value of the variable, NLSlots and log the slot name, value and confidence.

```
def slots = context.getVariable("NLSlots")

slots.entrySet().each
{
    entry ->
    sSlotName = entry.key
    sSlotValue = entry.value
    sSlotValue.entrySet().each
    {
        valueEntry ->
        value = valueEntry.key
        fConfidence = valueEntry.value
        context.log("Slot info: Name: " + sSlotName + " Value: " + value.getValue() + "
Confidence " + fConfidence)
    }
}
```

Integrating Intelligent Automation with Microsoft LUIS

Important
This page will cover information on deploying bots when Microsoft LUIS is the NLU provider.

Integrating Microsoft Bot Framework with Intelligent Automation requires an understanding of how LUIS and the Bot Framework work in tandem to provide an NLU service. Bots are created as code compared to a bot authoring environment like Dialog Engine or Google Dialogflow.

- Integration with Direct Line API 3.0
- Productised integration to MSFT
- Available for both Chat and Voice.
- This is not a direct integration with LUIS NLU
- Direct integration between Intelligent Automation to MSFT (No UniMRCP)
- Intelligent automation does not use Bot Gateway to integrate with AI systems, only to integrate with chat/messaging system
- For voicebots on MSFT it uses Google SST. (Intelligent Automation does not integrate with Microsoft cognitive services)
- Context is passed in on each turn and it is only a "one way" conversation. IA passes context to MSFT Bot Framework but not vice-versa.

To connect, you need separate credentials for LUIS and the Bot Framework.

Configuring LUIS

You can access LUIS at any of the following URLs depending on your region:

- US
- Australia
- Europe

You can find more information about Azure regions here.

Configure the credential mapping between LUIS and IA.

LUIS Credential	Intelligent Automation Configuration
Endpoint URL	LUIS Authoring Region
Primary Key	LUIS Subscription Key

Configuring Microsoft Bot Framework

The Microsoft Bot Framework is comprised of an open-source SDK and tools for end-to-end bot development. Microsoft Azure Bot Service helps in managing and deploying the bot. To integrate LUIS with the bot, you have to provide the LUIS credentials in the .env file of your bot.

Guidelines on Integrating IA with Microsoft Bot Framework

As the Microsoft Bot Framework is a toolkit for app developers, the following guidelines and assumptions are provided to ensure an optimal integration between Intelligent automation and the Microsoft Bot Framework.

An Activity is the primary means of information exchange between the Bot Framework and IA. The Activity schema is explained in detail. We will use the **value** field in the Activity schema to demonstrate how this field can be used in IA.

The following sections describe what IA expects from a bot in additional to some code samples (Node.js scripts)

Passing Context Variables

Context variables collected in a call-flow can be passed to the bot using the **Natural Language Settings** menu. The context parameters will be available in the *slots* sub-field in the *value* field of the activity.

```
Activity
{
    …
    ‘value’:
    {
        …,
        ‘slots’:
        {
            ‘var1’: 32
            ‘var2’: “Some string”
            ‘var3’: true
        }
    }
}
```

If you wish to leverage this information, it will be incumbent for the bot to extract out this information and make it persistent throughout the interaction You can now use extract this information and make it available to your bot to take action. The following code sample shows an additional step added when processing the first message of the conversation. It shows how incoming context variables can easily be read from the incoming message and stored in a persistent state.

```
/**
     * Pulls context variables out of the incoming message. These would have been
        * included in the message incoming from IA
     */
    async contextVarStep(stepContext) {
        let incomingActivity = stepContext.context['_activity'];
        let state = stateManager.fetchState(incomingActivity.conversation.id);
        if (incomingActivity.value && incomingActivity.value.slots) {
            state.contextVariables = incomingActivity.value.slots;
        }

        return await stepContext.next();
    }
```

The following sample shows how to persist the extracted information (the origin and the destination airports) so that additional follow-up questions are not required by the bot.

```
async actStep(stepContext) {
    …
    let state = stateManager.fetchState(stepContext.context["_activity"].conversation.id);

    let bookingDeails = state.contextVariables ? state.contextVariables : {};
    …
    // Call LUIS and gather any potential booking details. (Note the TurnContext has the
    //response to the prompt)
    const luisResult = await
                   this.luisRecognizer.executeLuisQuery(stepContext.context);
    switch (LuisRecognizer.topIntent(luisResult)) {
        case 'BookFlight':
            …
            // Extract the values for the composite entities from the LUIS result.
            const fromEntities = this.luisRecognizer.getFromEntities(luisResult);
            const toEntities = this.luisRecognizer.getToEntities(luisResult);
            …
            // Don't overwrite what we get from IA if we get nothing
            if (toEntities.airport) {
                bookingDetails.destination = toEntities.airport;
            }

            if (fromEntities.airport) {
                bookingDetails.origin = fromEntities.airport;
            }
            …
    }
    …
}
```

Intent Recognition

For intent recognition to work, the recognized intent has to be passed on to Intelligent Automation along with the conversation data.

The following example assumes that the intent received from LUIS is stored and persisted. It also includes the intent in a format that can be understood by IA.

```
async finalStep(stepContext) {
   …
   const currentIntent = stateManager.fetchSate(
        stepContext.context["_activity"].conversation.id)
        .currentIntent

   …
   // This is an activity sent at the end of the conversation.
   // will be detailed below
   let endActivty = {
        type: “endOfConversation”,
        value: {
            ‘currentIntent’: currentIntent
        }
   }
}
```

As long as the intent is included in the *endOfConversation* activity, it will be recognized and handled by IA. This allows the call-flow to follow any paths set in the **Intent Lists** menu of the relevant sub-modules.

If the intent is not included, IA will not display an error for the unrecognized intent and the call flow will branch off the call-flow defined for errors. It is also possible to return the intent before the the *endOfConversation* activity.

End of Conversation

As shown in the previous example, each conversation should include an activity of type *endOfConversation* along with any final messages. Again, this *endOfConversation* message must include the intent parsed by LUIS.

The following example demonstrates how to send the *endOfConversation* activity along with a final message. It would be placed within the *finalStep* method.

```
return await stepContext.context.sendActivities([msgActivity, endActivity]);
```

Introduction Messages

Due to an inconsistent timing behaviour in Bot Framework, currently IA will make the first move in terms of sending a query to the bot. As a result, we ask that messages not be sent from Bot Framework to IA until the user’s first query.

To provide an introduction message, use the **Initial Question Prompt** available within the **Other Prompts** section of the **Prompt List** menu within the appropriate sub-module.