Produktová dokumentace se přesouvá do modernější a rozšířené podoby na docs2.feedyou.ai. Stávající již nebude rozšiřována a časem bude nahrazena.

API - English version

- Integrations basics
- Export API
- Trigger API
- External API recommendations

Integrations basics

Integration is anything which enables chatbot to connect with outside world. It could be initiated by the chatbot using step on certain point in the chatbot tree or in the opposite by some external system by triggering some of bot APIs.

Integration step

Integration step could be added to any place of chatbot tree and is triggered in the moment when any user is passing given point in the conversation. It can call any of supported APIs, pass parameters inside (static or dynamic based on data we already know about the user using variables) and store results for further usage.

Use-cases

- Fetching some data from system to show (in mesage, carousel, ...) or use for further decisions in the bot flow (for example show available job positions) - examples of use:
  - Show carousel of available job positions in ATS system
  - Show carousel of recommended products in eshop
  - Get maximum loan value based on values entered by the user
- Storing some data from bot to external system (add new candidate to selected position using data user entered)
  - Store candidate application to ATS system with all gathered information (name, email, phone, position ID)
  - Order some selected product in e-commerce platform
  - Create support ticked based on gathered clues in servicedesk systems like Jira
- Triggering some action in external system (trigger remote modem restart in internet connection helpdesk bot)

Architecture

Every integration step type consists of two parts. First of them is JSON schema which describes what input parameters of which types are expected, which of them are required and what output structure should be expected. Second part is the code itself which implements:

- How is bot authenticated (in the most cases using some predefined token stored in the bot settings)
- How are parameters transformed and attached to the call

- How are output data transformed before saved into output variable
- Error handling
  - show fatal bot error to the user and stop conversation
  - trigger some predefined dialog
  - just log error and save some flag to user data variable so some decision could be made in the tree based on it

In ideal case, integration step performs just a single API call but in some advanced scenarios it is often needed to perform multiple calls and combine input/output data to fulfill desired goal.

Advanced topics

Timing

Integration step could behave in three different ways from the point of view of timing and performance.

- **Synchronous call**
  - User needs to wait for completion of API call during conversation
  - All data returned from API could be saved and used further
  - Data are not stored in bot database so filtering/searching should be performed on target system and just required subset of data should be returned
  - System should be stable and response should be fast (max. few seconds) to not let user wait too long
- **Asynchronous call**
  - Fire the request but do not wait for the result and continue immediatelly
  - Can be used to trigger some not-critical action where we do not need to know the result
  - For example track event into some analytics system
- **Call on background**
  - Data could be downloaded on background and stored into bot's database
  - They are later accessed locally by integration step during conversation quickly without need of any other system
  - Bot needs to filter/search data by itself which has limited performance (max. thousands of rows) and features (only filter by by tags or by locality radius)
  - Works also fine for slow and unstable data sources
  - There could be several ways how to trigger background data update
    - Periodical pull (reliable but consumes an unnecessarily large amount of resources)
    - Push (external server pushes data to the bot when some change occurs - always all of them or incrementally)
    - Pull on change (external server just signals bot that there is some change and bot do the same as during periodical pull)

Level of abstraction

It is very important to consider which level of abstraction use during the process of integration step design. For example imagine that we have some system called "MyCRM" with a lot of modules with REST API which can perform insert/update/delete operations on any of them when every module has different set of fields. Goal of prepared integration is just to add customer with given name and email. Now we have at least three levels of abstraction to choose from:

- **Action level**
  - The most specific, single purpose, very easy usage, need to change the code during every change of logic
  - Integraton called `mycrm-add-contract`
  - Parameters `name=...` and `email=...`
  - Returns `id=...` which could be used directly
- **Protocol level**
  - The most generic, versatile, difficult usage, do not need to change the code when logic changes, API needs to support such generic format
  - Integration called `rest`
  - Parameters `endpoint=mycrm.com/api/v1/customers`, `method=post`, `body={"name":"...","email":"..."}`
  - Returns `{"response": {"body": {"id":"...", ...}, ...}, ...}` which needs to be parsed
- **Service level**
  - Balance between two levels above
  - Integraton called `mycrm`
  - Parameters `module=customers`, `action=add`, `body={"name":"...","email":"..."}`
  - Returns `{"id":"...", ...}` which needs some simple parsing

It could not be said in a generic way which of these levels is correct. It always depends on given use-case, specs of used API and plans for future modifications of logic by non-programmers.

Channels

Channel is used to transfer messages from bot the user and back. We have some channel integrations built in the product itself, but it is possible to integrate any custom chat channel with proper API. There are three ways how to do it:

- Feedyou will implement connector to present API of the new channel
- Channel will implement connection to generic Microsoft Bot REST API which is already supported (more info here)
- Channel will be part of some channel-aggregation platform which is already supported by Feedyou such as MS Bot Service or Mluvii

Some chat channels also support transferring events during the same "pipe" which is used for messages. Example could be WebChat component which allows:

- during initialization of chat component on the web page, any data we know about the user (such as email, selected page, auth token, ...) could be passed in and then can be further used in the chatbot tree for decision or passed into other integrations (for example auth token passed from the web page could be used for API authentication with user-scoped access permissions)
- when any event happens for page (for example user clicks some element, fill form field, etc.), webchat can send event to the bot to save some user data, start selected dialog and so on
- in the opposite when user reaches some point in the tree, it is possible to send event from bot to the webchat component and trigger predefined listener to perform any code in the page (for example close chat, reload page, ...)

Dialog trigger

External system can trigger selected dialog for given user by calling Trigger API which has following parameters:

- Selected user address (probably previously stored in some output integration, some segmentation filter could be also available in the future instead of single ID)
- Dialog ID to trigger
- Optionally list of user data to preset before triggering dialog

You can read more in Trigger API documentation.

User data export

There is an API exposed by every running chatbot which could be used to retrieve all stored data for every user. This API is mainly used for analytic purposes (such as how many users have passed certain point in the tree - and thus have given storage filled) but also could be used for anything else. There are available basic parameters that could be used to filter out only given subset of users.

You can read more in Export API documentation.

Preset integrations

There is already a lot of integrations already implemented in Feedbot Designer. You can check this spreadsheet to find out more. It is important that every system could be integrated in large number of different ways with different options so it is needed to consider given use-case carefully.

API automation platforms

Not all integrations needs to be programmed directly in the bot's code. Different kinds of automation/integration platforms or tools could be used to connect different services including bot to implement needed logic. Good examples of these platforms could be:

- Integromat
- Zapier
- Microsoft Azure Logic App
- ... and many others

Adding new integrations

Following list tells what is often needed to have to allow new integration to be implemented:

- Well described use-case incl. the way how integration should be used in the chatbot tree such as input/output variables and how they will be used on other parts of the tree
- API specification incl. authentication (if target system currently do not have any and is going to prepare some just for the chatbot, take a look at the example how we think this API should look like)
- Example credentials to be able to test integration during implementation against real system

Export API

> Full and up-to-date version of the whole Management API including export module could be found in our Swaggerhub.

Base URL for export API is bot-specific `https://feedbot-${BotId}.azurewebsites.net`. Please use the `?code=...` query param or `x-functions-key: ...` HTTP header to perform token based authentication.

Get users data

> Calling this endpoint without timestamp filters can return very large amount of data and affect bot performance. Regular daily fetching of data from the past day is recommended approach.

`GET /api/management/export/userData`

Returns list of objects containing all user-scoped data which bot persists. Optional filters parameters:

- `notEmpty` use comma separated list of fields (in *camelCase* format) which should not be empty to include given user in the output list
- `fromTimestamp` and `toTimestamp` second-based UNIX timestamp to limit users in the output based on the date of last message
- `where...` where `...` is name of field which should be equal to the parameter value

`userId`
`timestamp`
`... all storages in camelCase ...`

Examples

- `GET https://feedbot-${BotId}.azurewebsites.net/api/management/export/userData?notEmpty=phone&fromTimestamp=1559901734` to get data of all users who have interacted with the bot after `06/07/2019 10:02:14` and have passed through question with `phone` storage
- `GET https://feedbot-${BotId}.azurewebsites.net/api/management/export/userData?whereGender=male&&code=${FunctionKey}`

to get data of all male users

Get users data snapshot

> This API method is available since bot hosting version v1.7.474

`GET /api/management/export/userDataSnapshot`

Returns list of objects containing all user-scoped data snapshots (generated by the "clear user data" action) which bot persists. Usage and all params are the same as in "Get users data" API.

`userId`
`timestamp`
`... all storages in camelCase ...`

Get NLP logs

> This API method is available since bot hosting version v1.7.474

`GET /api/management/export/nlpLog`

Returns list of objects containing all inputs entered by the user which was not possible to process using conversation tree. Desired time range could be specified using `fromTimestamp` and `toTimestamp` second-based UNIX timestamp.

`processed`

Get outgoing emails

> This API method is available since bot hosting version v1.7.474

`GET /api/management/export/outgoingEmail`

Returns list of objects containing all emails sent by the chatbot incl. recipient, subject, body. Desired time range could be specified using `fromTimestamp` and `toTimestamp` second-based UNIX timestamp.

`userId`
`timestamp`
`body`
`from`
`recipients`
`replyTo`
`stepId`
`subject`

Get outgoing SMS

> This API method is available since bot hosting version v1.7.474

`GET /api/management/export/outgoingSms`

Returns list of objects containing all SMS messages sent by the chatbot. Desired time range could be specified using `fromTimestamp` and `toTimestamp` second-based UNIX timestamp.

`link`
`message`
`provider`
`recipients`

Get fuzzy question answers log

> This API method is available since bot hosting version v1.7.474

`GET /api/management/export/fuzzyQuestionAnswerLog`

Returns list of objects containing all utterances processed by the fuzzy question answer plugin. Desired time range could be specified using `fromTimestamp` and `toTimestamp` second-based UNIX timestamp.

`userId`
`timestamp`
`intent`
`score`
`text`
`stepId`
`dialogId`
`matched`
`otherClassifications`

Get integration log

> This API method is available since bot hosting version v1.7.474

`GET /api/management/export/integrationLog`

Returns list of objects containing all integration invocations. Desired time range could be specified using `fromTimestamp` and `toTimestamp` second-based UNIX timestamp.

`userId`
`timestamp`
`url`
`method`
`requestBody`
`requestHeaders`
`responseStatus`
`responseBody`
`duration`

Get URL tracker

> This API method is available since bot hosting version v1.7.474

`GET /api/management/export/urlTracker`

Returns list of objects containing row for every click on URL button in bot by every user including target URL and domain. Desired time range could be specified using `fromTimestamp` and `toTimestamp` second-based UNIX timestamp.

`userId`
`timestamp`
`domain`
`url`

Trigger API

The Trigger API can be used to invoke a specific dialog for a selected user in a virtual assistant initiated by an external system.

Each bot runs on its own domain (usually in the format `https://feedbot-${BOT_ID}.azurewebsites.net`) and is protected by a key in either `?code=...` URL query parameter or `x-functions-key: ...` HTTP header.

`POST /api/management/trigger/${ADDRESS_JSON_AS_BASE64}/${DIALOG_ID}/${MODE?}`

- call parameters are:
  - user address (object with properties `bot`, `user`, `channelId` a `conversation?`) in JSON string format encoded with BASE64
  - ID of the dialog to be run
  - optional start mode, currently supported only by "clear", which first deletes the user's position in the tree and then starts the dialog
- the request body can contain a JSON object with additional information that is stored in the user's data
- the response to it needs to be defined in the Designer in advance, where additional information can be used using variables

Example of use

- the user requests a transfer of the credits on their loyalty card to their account, which may take several minutes
- the virtual assistant uses an integration step to call the customer's system, including passing the user ID to initiate the transfer
- informs the user that the transfer has started + triggers a timeout for 10 min, which would inform the user that the transfer has failed to complete if no response is received from the system by that time
- if the transfer was successful, the system calls the Trigger API POST `POST /api/trigger/5ca39df/transfer-complete?code=...` on the bot with the transaction details as JSON body of the request, which triggers a dialog that confirms to the user that everything was successful and displays the transaction details
- if the transfer has failed, the system calls the Trigger API POST `POST /api/trigger/5ca39df/transfer-error?code=...` on the bot. and the virtual assistant responds with an error message with instructions on how to proceed

External API recommendations

It is possible to prepare any number of "integration steps" in the virtual assistant, which are points in the conversation tree where the virtual assistant uses the API to integrate with another system in order to obtain some data necessary for its function or, conversely, to send some data that it has detected from the user to the system.

The virtual assistant can adapt quite a bit to the format and structure of the API, but if the API is created for the sake of the virtual assistant, the following recommendations can be followed to make the implementation as smooth as possible.

Recommended API properties

- Authentication is ideal in one of the following ways
  - username+password as a BASE64 string in the header `Authorization: Basic ...`
  - token in the header `Authorization: Bearer ...`
    - can be predefined for the entire virtual assistant
    - or if the bot e.g. runs inside the system where the user logs in, this token can always be passed to the bot with the scop of the user, so that the bot will have at most his permissions and not more
  - standard OAuth2 `client_credentials` grant (i.e. a separate request to obtain and extend the access token), which is then used in `Authorization: Bearer ...` with each request
- Ideally follow the REST rules for the HTTP methods used
  - **GET** for data collection
  - **POST** to create a new record or start an action (not idempotent)
  - **PUT** to update an existing record (its identifier ideally as a URL parameter)
  - **PATCH** to partially update an existing record (its identifier ideally as a URL parameter)
  - **DELETE** to delete an existing record (its identifier ideally as a URL parameter)
- Request and response body as JSON
- If the method requires it, implement filtering and sorting as URL parameters or as custom `X-...` HTTP headers
- It would be a good idea to keep the maximum response time under 3 seconds, because in some use-cases the API may be called synchronously in real time and the user will wait for the result

Sample

The virtual assistant can use the API to retrieve a list of scheduled events from the dependent system. The user IDs are passed, for example, to a WebChat component inside an internal system where the user is already authenticated. The data is used by the bot to display a "carousel" of scheduled events, where the user can select one of them and continue with some other action.

```
GET https://somedomain.cz/api/v1/user/33/events

Authorization: Bearer ...
Content-type: application/json

→ 200 [{"id":"1", "name":"Onboarding meeting", "date":"2020-05-07T08:22:30.871Z"}]
```

At the end of the communication, the virtual assistant can send information about the newly acquired user to the CRM. The request can include all the data that the virtual assistant has collected about the user.

```
POST https://somedomain.cz/api/v1/user

Authorization: Bearer ...
Content-type: application/json
{"email": "user@example.com", "name":"Jack", "surname":"User", "interestedInProductIds":
[244, 234]}

→ 200 {"id":"433"}
```