SYMPHONY

Developer Newsletter

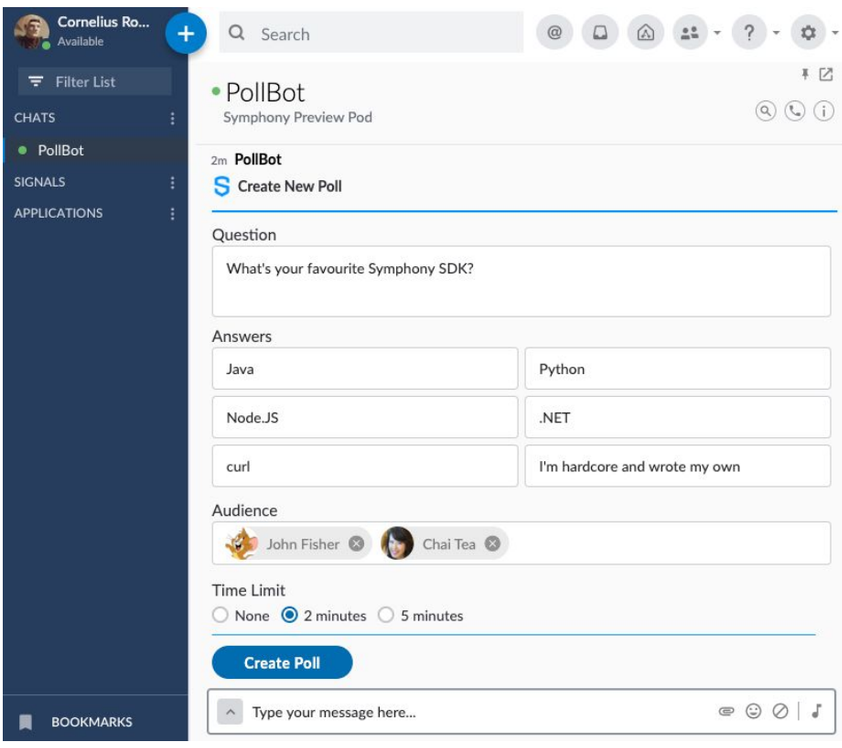
This newsletter covers bot examples built with Symphony Elements, a recap of Symphony Innovate 2019 in New York, the Innovate Hackathons and more.

Symphony Elements - Bot Examples

To demonstrate the capabilities of Symphony Elements, two bot examples were developed and published as open source examples for the community. These bot examples can be used as-is or as a reference to be customised for your organisation's needs. Both bots showcase various Elements in their respective workflows, introducing rich interactivity without requiring use of the Extension API.

1) Develop: Poll Bot (Java)

Poll Bot allows users to create and vote in polls. Simply fire a **/poll** command to request a poll creation form, fill in your poll questions, answer options, and a time limit, as necessary. These polls can be sent into a room for room members to vote on, or sent privately to a list of people via 1-to-1 messages with the bot. People then vote in the poll simply by clicking on the button with their chosen option. The bot collates all answers and waits for the poll creator to issue a **/endpoll** command or for the time limit to expire. Results are then displayed as a bar chart in the room or in a 1-to-1 message for private polls.



Creating a New Poll with the Poll Bot

This bot was built using Spring Boot, Symphony's Java SDK, and a MongoDB database to persist the poll and vote data. To dynamically construct Symphony Elements in MessageML, this bot uses FreeMarker templates and Entity JSON data payloads that represent the situation. For example, the poll creation form can provide 12 instead of 6 options by issuing a **/poll 12** command and the markup is generated dynamically using the **<#list>** FreeMarker tag.

```
<#list (1..entity["poll"].count)?chunk(2) as row>
  ...
  <text-field name="option${option}" placeholder="Option ${option}" ... />
</#list>
```

The data access layer is built using Spring Data Mongo and wired with repository interfaces that describe the dataset requested without requiring the developer to write the actual implementation. One example is **`findAllByCreatorOrderByCreatedDesc()`** that gets all polls sorted by created date for a specific user. Collating results leverages MongoDB's aggregation feature such that processing is offloaded efficiently from the bot onto the data source instead.

Get started with Poll Bot by visiting its **GitHub repository** today.

2) Develop: Expense Bot (Python)

Expense Bot enables users to create expense reports and subsequently add, remove, and submit expense reports to their managers. Expense Bot uses Symphony Elements, allowing users to submit expenses to the bot through interactive forms, dropdown menus, person selectors, and buttons. It captures the data submitted in these forms inside the **`action_processor.py`** file, taking advantage of the Python SDK's **`SymElementsParser()`** class to extract data submitted with Symphony Elements quickly and easily. It also leverages AWS Textract, allowing users to upload receipts as attachments and extract the relevant entity data belonging to this expense. Using Boto, the AWS SDK for Python, developers can easily create, configure, and manage AWS services. After authenticating to AWS, users can extract entities such as date, organization, or price for a given receipt:

```
comprehend = boto3.client('comprehend', AWS_CREDENTIALS)
entity = comprehend.detect_entities(LanguageCode="en", Text=text)

if entity.get("Type", "") == 'DATE':date = entity.get("Text")
if entity.get("Type", "") == 'ORGANIZATION': description = entity.get("Text")
if entity.get("Type", "") == 'QUANTITY': quantity = float(entity.get("Text"))
```

SYMPHONY

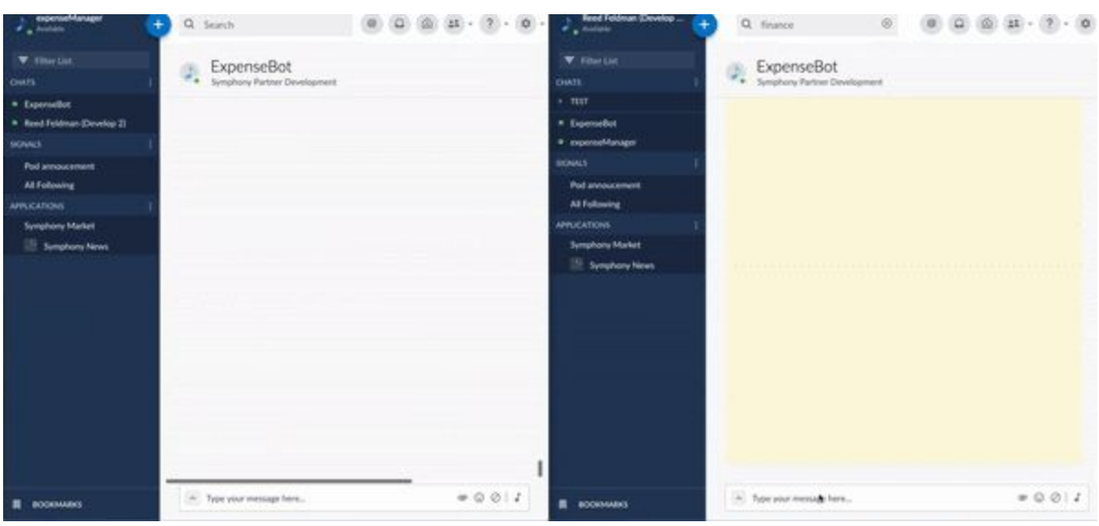
Developer Newsletter

To dynamically update the expense table in Symphony, Expense Bot uses **Jinja**, a modern templating language for Python. When a new expense is added, Expense Bot requests data from a MongoDB database for a unique userID and renders the expense table by passing this data as a json object to the associated Jinja Template:

```
from jinja2 import Template

expense_data = ExpenseReport.objects(owner=str(user_id), open=True)
with open(path_to_html_form) as file:
        template = Template(file.read(), trim_blocks=True, lstrip_blocks=True)
    html = template.render(expense_data)
```

Watch an example of ExpenseBot in action:



Read the **full documentation** of ExpenseBot to learn more.

Recap: Symphony Innovate 2019 Hackathons

Symphony hosted three developer Hackathons this year in Paris, London and New York surrounding Symphony Innovate 2019. Developers were given access to an environment and SDKs that allowed for a sneak peak of Symphony's new interactive Elements feature.

We also distributed our first Symphony Developer Survey to all Hackathon attendees. We received 25 responses across the regions! We thank the Symphony developer community for their participation in our developer events, and for providing us with invaluable feedback as we strive to improve our developer experience. We had members of Product Management, Platform Solutions, and API Support available in all regions to help out and listen to your feedback.

SYMPHONY

Developer Newsletter

A total of **25 teams** competed at the Hackathons: **7** teams in New York (hosted by **Citi**), **13** teams in London (hosted by **Credit Suisse**), and **5** teams in Paris (hosted in a charming art gallery near the Louvre). Here are all the winning developments:

Most Cutting-Edge Technical Development Award

- New York: BNP Paribas, Sales Workflow Bot – Maestro
- London: Team Reason, Between the Lines
- Paris: Société Générale, SG Bot, Structured Product for Clients

Most Progressive Business Workflow:

- New York: BNY Mellon and Blackrock, Legal Entity Identifier (LEI) eXchange Protocol (LEIxP)
- London: Credit Suisse 2, KC: The KYCD Status Update Bot
- Paris: Finastra, Client Confirmation Process

Read more about the winning projects on our **blog**.

Symphony Innovate 2019

At Symphony Innovate 2019, Symphony partnered with AWS to demonstrate how to build a support ticket notification system within Symphony on top of the newly released AWS EventBridge product.

EventBridge works by exposing the CloudWatch events from a partner SaaS application, from which you can build Lambda expressions that trigger on the events coming across the wire. In this case, we built a Symphony integration with Zendesk that loads the Python SDK into a Lambda expression that triggers when an event that corresponds to a new ticket or a tag being added comes across the EventBridge. The Lambda expression parses the incoming event, and then sends a message to a different Symphony chatroom based on the context of the ticket, sender and associated tags.

Watch a **presentation** for this integration (includes example code).

SYMPHONY

Developer Newsletter

Developer Events

Make sure to join one of our **Symphony Developer Meetup Groups** to receive updates on upcoming developer events near you!

Share with a Colleague

Know a colleague that will find the developer newsletter useful? Help them **subscribe to the newsletter** now.