Dedoctive Tutorials

This introduction provides an overview of **Dedoctive**, a hybrid platform designed to bridge the gap between structured business processes and autonomous artificial intelligence.

Overview of Dedoctive

Dedoctive is a hybrid workflow environment that combines the predictability of **BPMN (Business Process Model and Notation)** with the adaptability of **Agentic AI**. While traditional BPMN systems offer easy-to-understand, fixed processes, they often lack the flexibility to adapt to changing goals. Conversely, Agentic AI can solve problems independently using various tools but can be unpredictable and may lack the ability to request user input mid-process.

Dedoctive combines these approaches, allowing for:

- **Hybrid Workflows:** Blending structured steps with flexible, AI-driven decision-making.
- **Interactive Agents:** Enabling agents to request real-time information from users via forms.
- **Unified Design:** Managing both traditional and AI-driven processes within a single environment.

The Development Environment

The tutorials utilise the **Dedoctive Designer**, an IDE used to create, test, and debug "Deduction" workflows. These workflows leverage several key technologies, including:

- **Google ADK (Agent Development Kit):** Used as the underlying framework for agents.
- **Model-Context-Protocol (MCP):** A standard for accessing external tools and servers.
- **LiteLLM:** A proxy server used to access a wide range of Large Language Models (LLMs).

Scope of the Tutorials

The provided documentation guides users through a progression of increasingly complex tasks:

- **Foundational Skills:** Creating basic BPMN workflows with **script tasks** and visual debugging.
- **Human-in-the-Loop:** Designing and embedding **JSONSchema-compliant user forms** to collect and manipulate data.
- **Agentic Orchestration:** Integrating **LLMAgents** that can non-deterministically call external MCP tools to satisfy user requests.
- **Modularisation and Recursion:** Developing **sub-workflows** and external processes that can be invoked as tools, allowing for complex, hierarchical, and even self-recursive business logic.

- **Advanced Data Handling:** Using **workflow-global variables** (`workflowData`) and dynamic form elements to manage large-scale data processing.

Unique Features

A standout feature highlighted in these tutorials is **Dynamic Debugging**. This allows developers to 'rerun' a workflow while only executing tasks that have changed, reusing prior results for unchanged steps. This significantly reduces the time and cost associated with debugging long-running or expensive agentic tasks.

Ultimately, these tutorials demonstrate that the perfect balance for automating complex tasks lies between **deterministic BPMN** and **ad-hoc agentic flows**, creating what the sources describe as "the best of both worlds".

Table Of Contents

Dedoctive Tutorials

Background

Work processes are often complex, so they're typically mapped using **BPMN (Business Process Model and Notation)** — a standard way to visually describe workflows. Some systems can even run these diagrams directly, following each step exactly as defined.

- **Pro:** Easy to understand and explain.
- **Con:** The process is fixed and predictable — it can't adapt on its own.

Newer **Agentic AI** systems work differently. You describe the goal and give the AI tools to use, and it figures out how to solve the problem by itself.

- **Pro:** You can describe tasks in plain English.
- **Con:** The AI's approach changes from run to run and can't ask users for extra input mid-process.

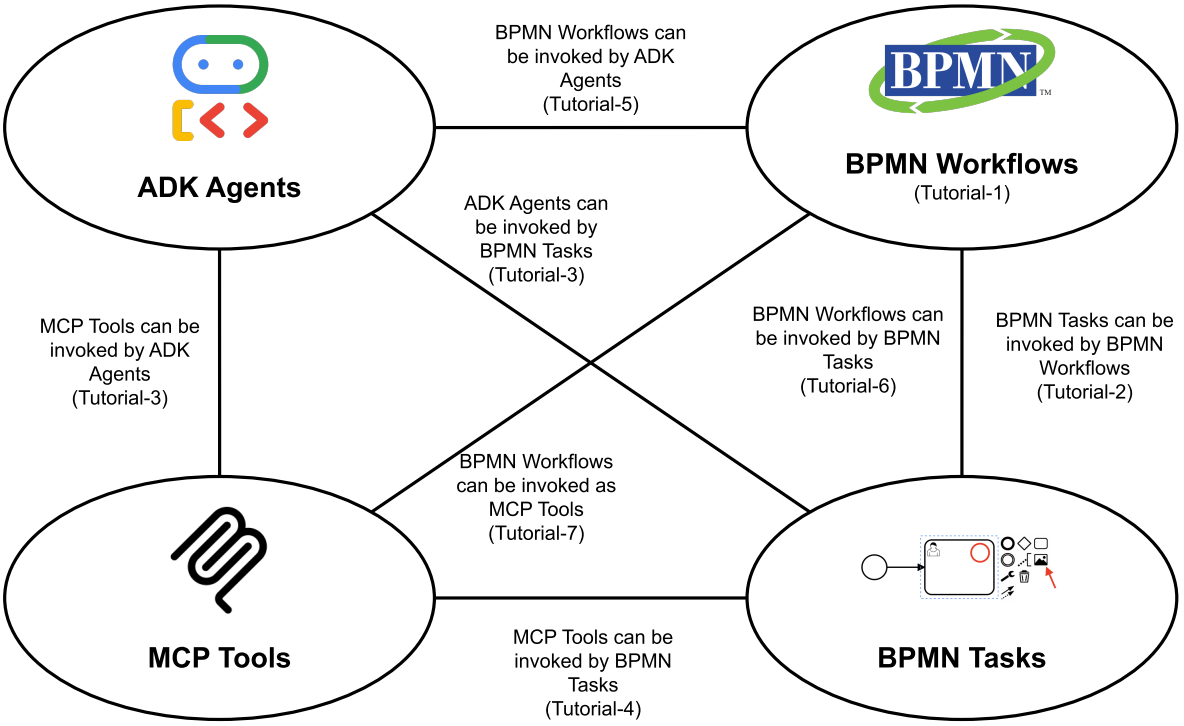
Dedoctive combines the best of both worlds:

- **Hybrid workflows:** Mix BPMN for structured steps with agentic AI for flexible decision-making.
  Benefit: Get both predictability and adaptability.
- **Interactive agents:** Agents can ask users for more information using forms.
  Benefit: Smarter, more accurate outcomes based on real-time user input.
- **Unified design:** Manage traditional and AI-driven processes in one environment.
  Benefit: Easier to design, maintain, and evolve workflows.

Developing Deduction Workflows

It is assumed that you have some familiarity with BPMN workflows. If not this video is a great introduction: The Only BPMN Tutorial You Will Ever Need To Watch (For Beginners) This video uses a different tool than Dedoctive Designer for developing the BPMN, but the principles behind BPMN are identical. Our tutorials will all use Dedoctive Designer to design and debug the Deduction workflows.

It is also assumed you have some familiarity with Google ADK (ADK Agents) and Model-Context-Protocol (MCP). What is MCP? Is a good explanation to start. Have a look at the introductory sections of the Google Agent Development Kit as a start on Agents.



1. Tutorial-1

1.1. Scope: Run Designer to create and run a workflow that executes a simple script

This first tutorial is designed to illustrate creating a BPMN, adding a simple script task, and then debugging to view the results.

Dedoctive Designer is the IDE for Dedoctive Deductions. It allows Deduction BPMN workflows to be created interactively, and tested using the DedoctiveDeduction server to execute the workflow.

As these tutorials will demonstrate, these Dedoctive Deduction workflows can include:

- BPMN tasks, such as scripts, and events
- User Input forms, to collect input from users
- Sub Workflows, to execute a sub workflow
- External workflows, to execute a workflow that is defined in an entirely independent BPMN file
- mcpTool tasks to make request to external mcpServers
- LLMAgent tasks to perform Agentic processing, including accessing mcpServers, and invoking workflow agents and user agents

This allows the full spectrum of agentic through to deterministic processes to be designed and tested within a single IDE.

Once the workflow has been deployed, Dedoctive UI is the production-ready tool from which authenticated and accredited users can launch these workflows. However, for now we will not use the Dedoctive UI

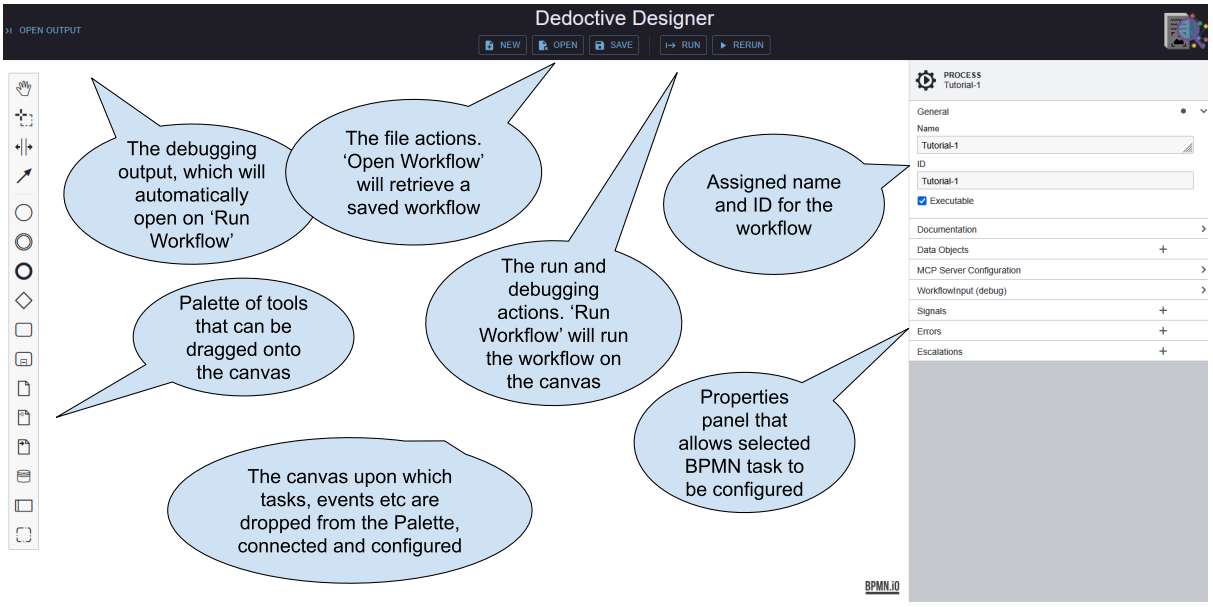
The BPMN for this tutorial can be found in Tutorials/Tutorial-1

1.2. Steps

Navigate to localhost:4000 to launch DedoctiveDesigner

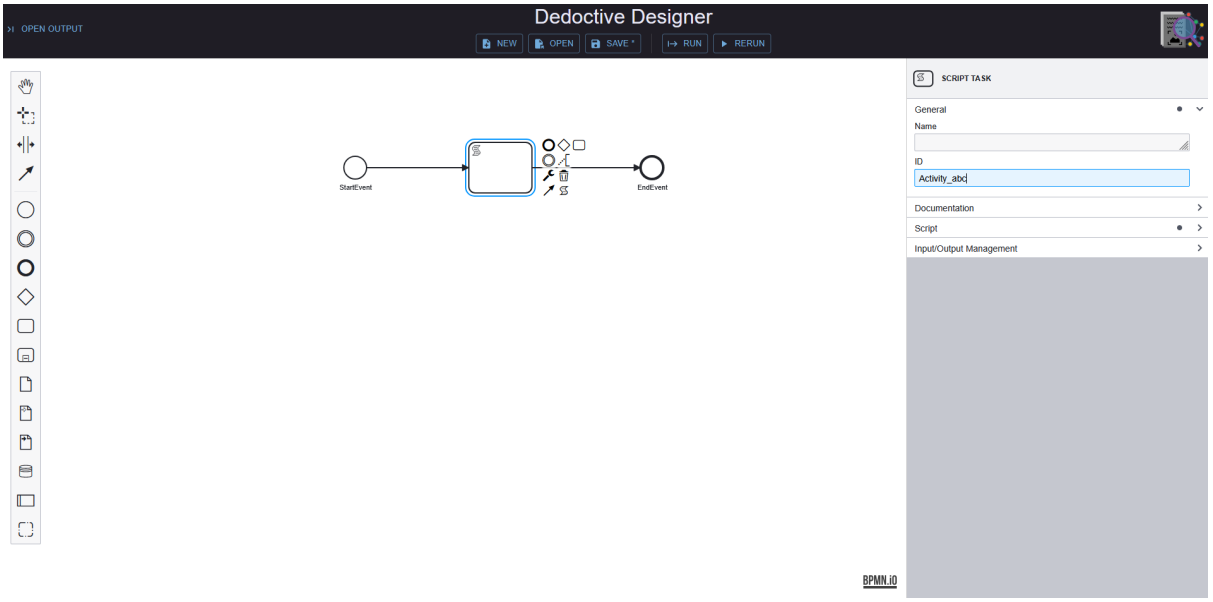
After logging in with your Google account, start by clicking “Start a new workflow” and enter the name Tutorial-1.

The below image provides an overview of the Designer user interface:

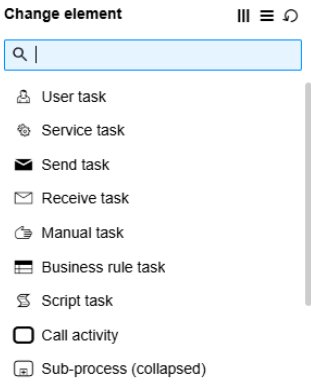


It is a good discipline to name the overall process, and optionally assign an ID instead of accepting the system generated ID. This is useful for later debugging purposes.

1. All workflows need at least one start and stop event.
    a. Drag these from the palette onto the canvas.
    b. It is a good idea to connect them with a connector.
    c. It is also useful to provide names that suggest what triggers the start event and what has been accomplished by the time of the end event as well as the auto-generated IDs for each of the components added to the canvas.
    d. For example the start event is labelled ‘StartEvent’ with the same ID. Similarly for the end event.
2. Next drag a ‘task’ from the palette and drop onto the connecting line. Designer will automatically insert it into the connection between the start and end events.
    a. Note that the task at this stage is anonymous.
    b. Selecting this new task will change the Property panel, but also a task context menu as shown below:

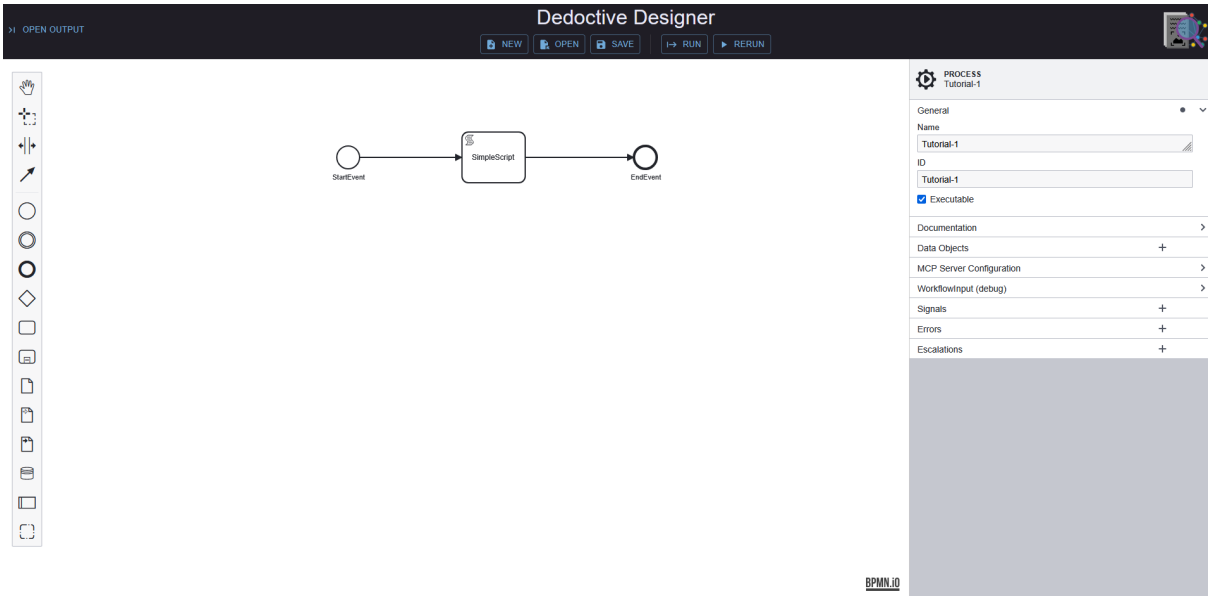


3. Clicking on the ‘spanner’ will show a list of options for the task

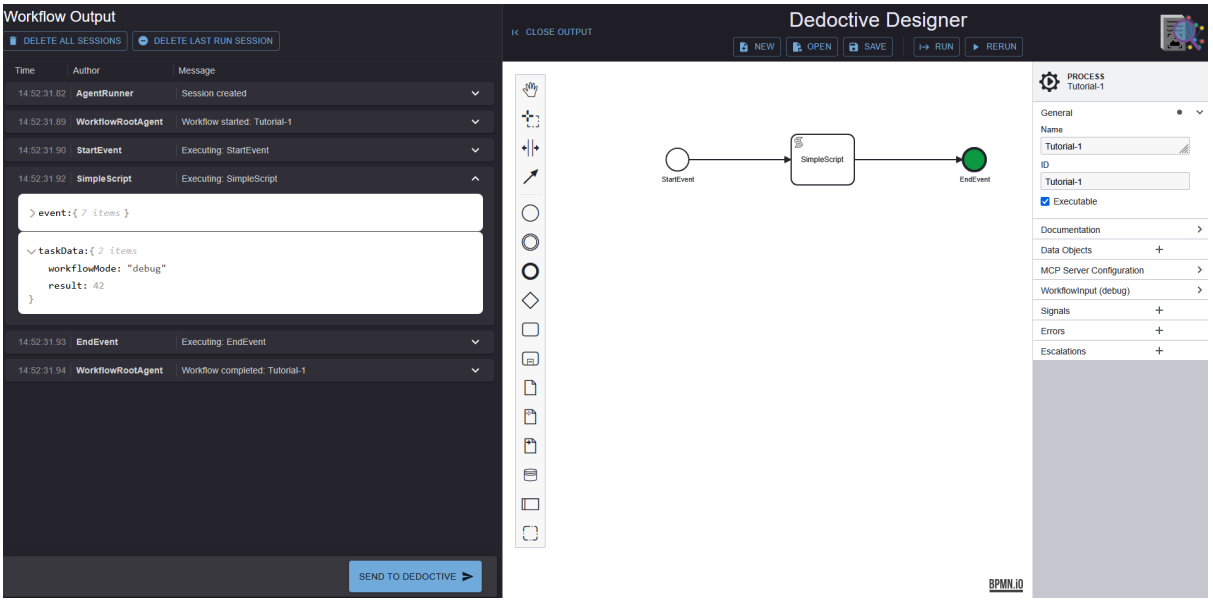


   a. Select Script task
   b. Assign a name and ID in the Property panel, ‘SimpleScript’
   c. Select the ‘Script’ section and add some Python, in this case:

```
result=42
```



4. Run this workflow and inspect results
    a. Navigate to Run Workflow to launch and debug
    b. It is not necessary to save the workflow before running, but a good discipline to do so.
    c. The Workflow Output panel will automatically open up and show the results of the workflow execution.
    d. The components of the workflow on the canvas will also be highlighted as they are executed. In this case it is unlikely you will see them change as it all happens so quickly, leaving only the StopEvent highlighted.
    e. The Workflow Output panel shows Time/Author/Message associated with each event when running this workflow
    f. Many of these events are system-generated:
        i. The AgentRunner indicates the initialization of the Agent that is the 'host' of this workflow. This is a Google-ADK agent
        ii. The WorkflowRootAgent indicates the start of the BPMN workflow, which we named Tutorial-1
        iii. StartEvent shows that the engine has initiated and started the workflow
        iv. SimpleScript is the Script task that we created
            1. Expanding the SimpleScript reveals the 'event' and 'taskData'
            2. The 'taskData' is a dictionary to which tasks, in particular scripts, will add their results. Additionally tasks can access the data within this dictionary.
            3. We can see the 'result:42' has been automatically added to the dictionary.



    v. The EndEvent is the end event from the workflow
    vi. WorkflowRootAgent signals the completion of the agent.

1.3. Takeaways

1. Workflows can be easily assembled as a flow of tasks

2. When a BPMN workflow is launched, what is happening is that a WorkflowAgent, one of the subClasses of DedoctiveAgent which in turn is a subClass of the Google ADK BaseAgent, is run using agent-runner code
3. The ADK agent-runner code for a workflow, will load the appropriate BPMN and execute the tasks within that workflow.
4. The events that appear in the Workflow Output panel correspond to the ADK runner events.

2. Tutorial-2

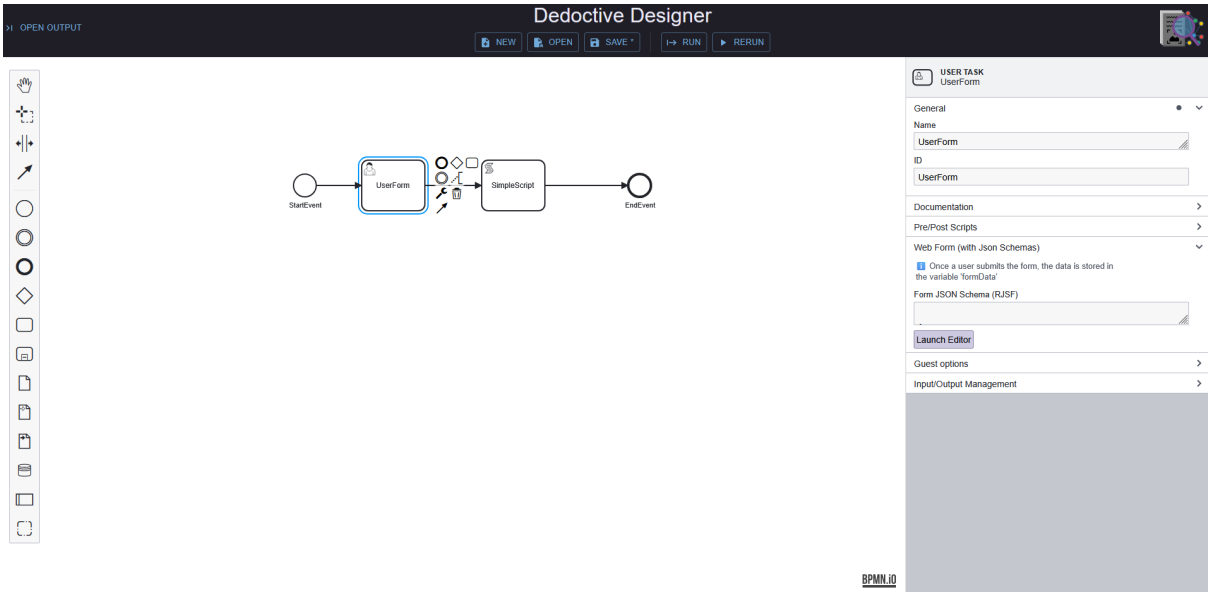
2.1. Scope: Add a data entry form to the workflow, and use the returned values in a script task.

This tutorial builds on the Tutorial-1 showing how a user form can enter some data, a script can access that data, and another script can manipulate that.

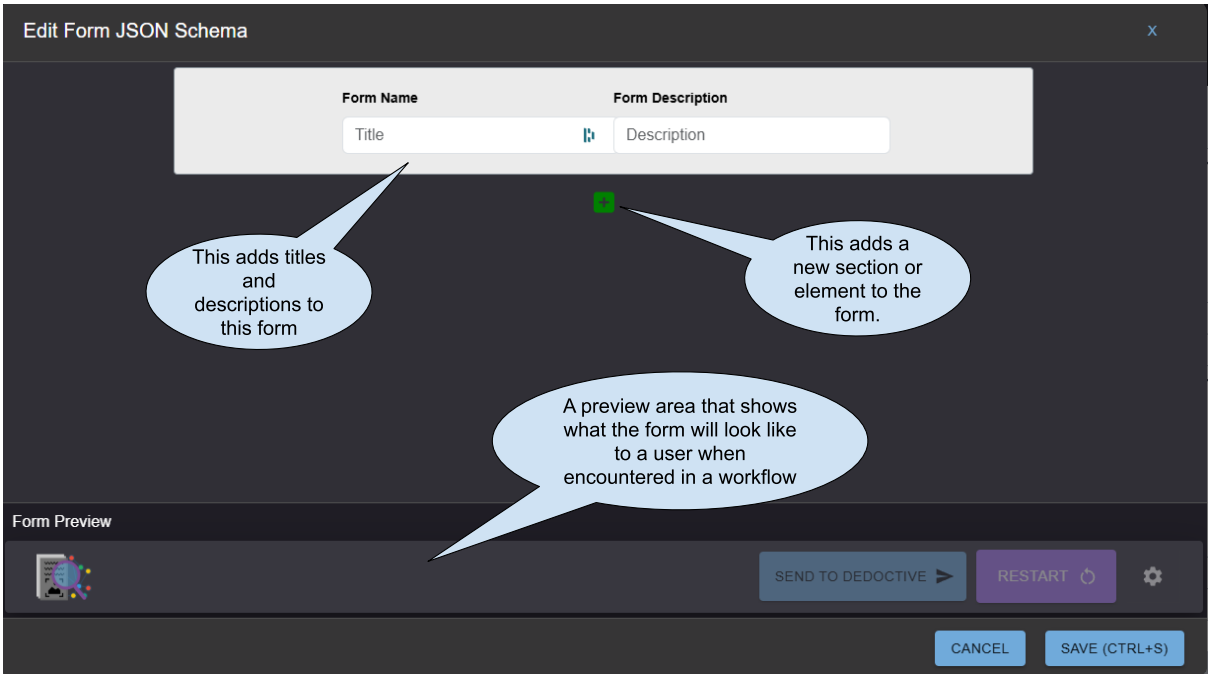
The BPMN for this tutorial can be found in Tutorials/Tutorial-2

2.2. Steps

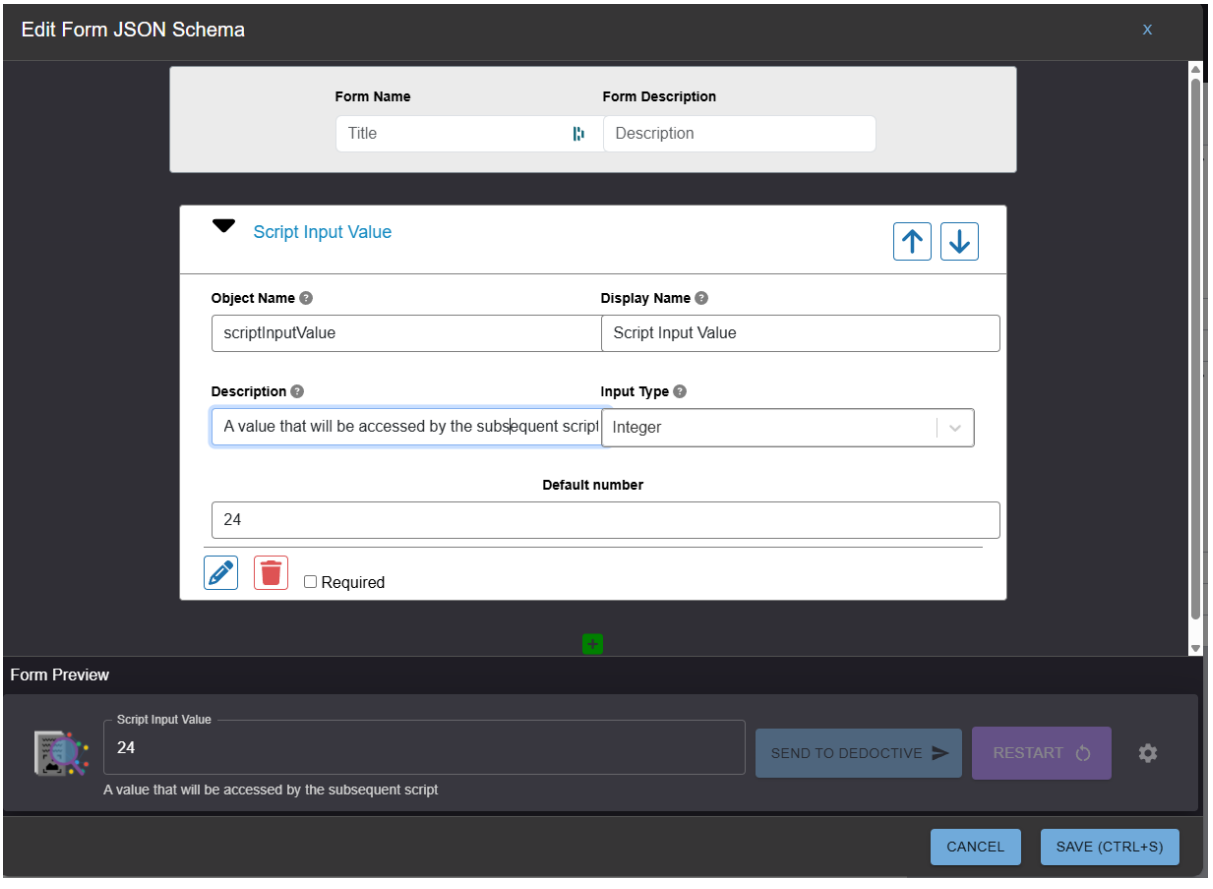
1. Starting with Tutorial-1 BPMN, add a task before the script and configure it as a User Task.
    a. Drag an anonymous task from the palette and drop onto the connection between the StartEvent and the SimpleScript task.
        i. This automatically inserts into the connected between these two components
        ii. You do not have to do it this way. You can add, realign, and delete connectors.
    b. Use the spanner on the new task's context menu to show a list of task types.
        i. Select User Task for the type
    c. Name the user task 'UserForm' in the Property Panel



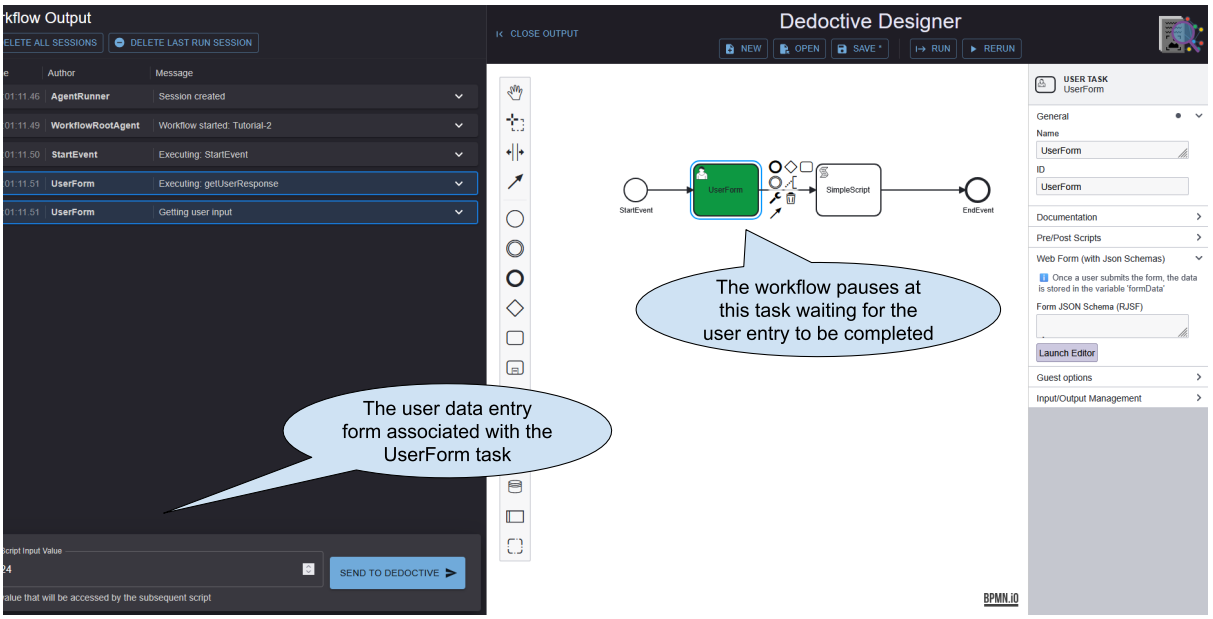
2. In the Property Panel on the right, select the Web Form section, and click 'Launch Editor' to show the JSONForm editor



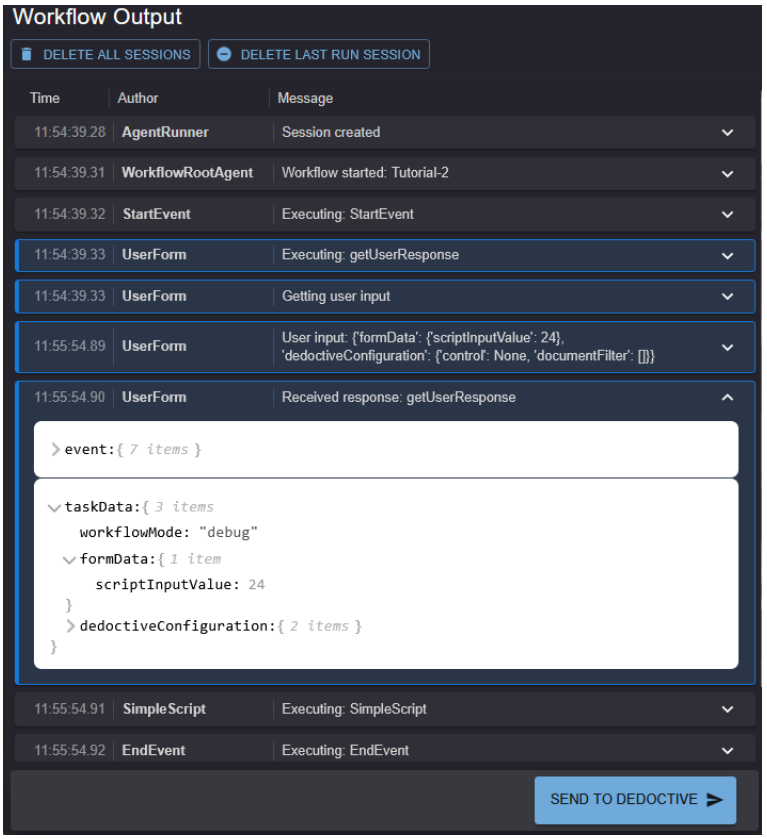
3. Add an integer element that will be modified by the subsequent script task
    a. Click the “+” button, choose “Form element” and click the arrow to expand
    b. Details for the element are shown in the image below
    c. After completing, ‘save’ to the BPMN workflow
    d. Note that the JSONSchema form definition is embedded within the workflow BPMN (aka XML) file
    e. The JSON of the schema is shown in the text window if you really want to see it or even edit it (at your own peril!)



4. Run this workflow
    a. Instead of running to completion the workflow pauses at the UserForm task, and the data entry form appears at the bottom of the Workflow Output panel



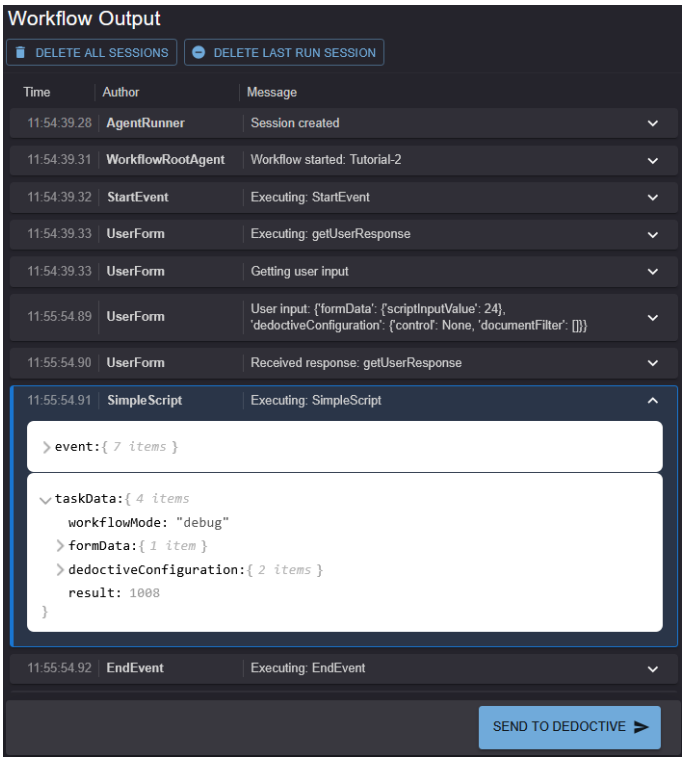
    b. Change the Script Input value (to say 24), and then ‘Send to Dedoctive’
    c. The workflow runs to completion, so where has this value gone? It now appears as an entry in the ‘taskData’ dictionary, named formData. ‘formData’ is the name assigned by the JSON schema. The scriptInputvalue is an entry in this dictionary.



    d. We can access any of the ‘taskData’ dictionary entries in any subsequent task. For example we could change the SimpleScript to:

```
result=42* formData.get('scriptInputValue',0)
```

e. Then when we inspect the Workflow Output for the SimpleScript task, we see the modified result value



2.3. Takeaways

1. User data entry forms, complying with the JSONSchema standard, can be defined and embedded within the BPMN
2. The workflow will pause whilst the user can submit a response to the form.
3. The ‘results of any form are within the taskData dictionary, as element ‘formData’
4. This will contain whatever is assembled in the user defined form.
5. The values of the formData dictionary are accessible in subsequent tasks, such as scripts
6. Selecting a task within the BPMN canvas will highlight the events corresponding to that task in the Workflow Output panel

3. Tutorial-3

3.1. Scope: Add an agentic task to the workflow which can access external MCP tools.

The previous Tutorials have demonstrated the deterministic nature of Dedoctive’s Deduction workflows. However, creating a deterministic workflow to multiply two numbers is neither challenging nor ‘agentic’. Agentic is a process in which the calling of tools (aka performing tasks if using BPMN terminology) is handed over to a Large Language Model (LLM). The LLM is given instructions, and which ‘tools’ are available for it to use in fulfilling those instructions.

The BPMN for this tutorial can be found in Tutorials/Tutorial-3

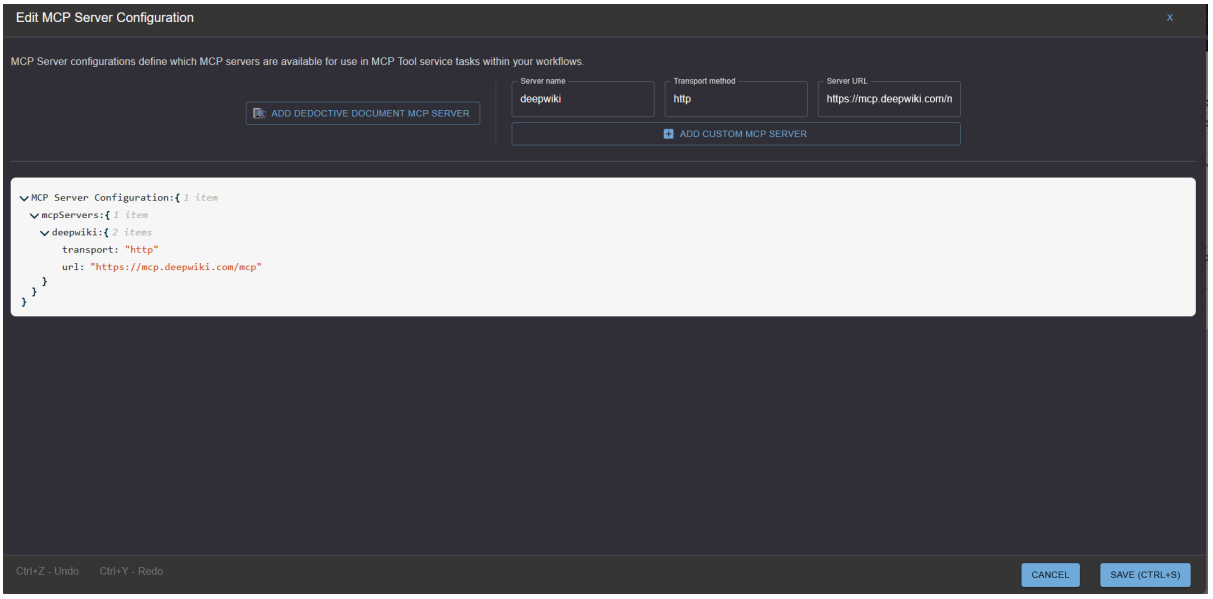
3.2. Steps

An LLMAgent is another type of task that can be included in a workflow. This LLM agent can perform the ‘agentic’ processing required for non-deterministic workflows.

Notes for Dedoctive Developer Edition:

- *API keys for LLMs must be entered into the .env file during setup to be able to select them in the Designer for the LLM Agent*
- *The DedoctiveDocument MCP server is not available in the Developer Edition, please enquire if you are interested in the capabilities of the knowledge model for complex analysis of data*

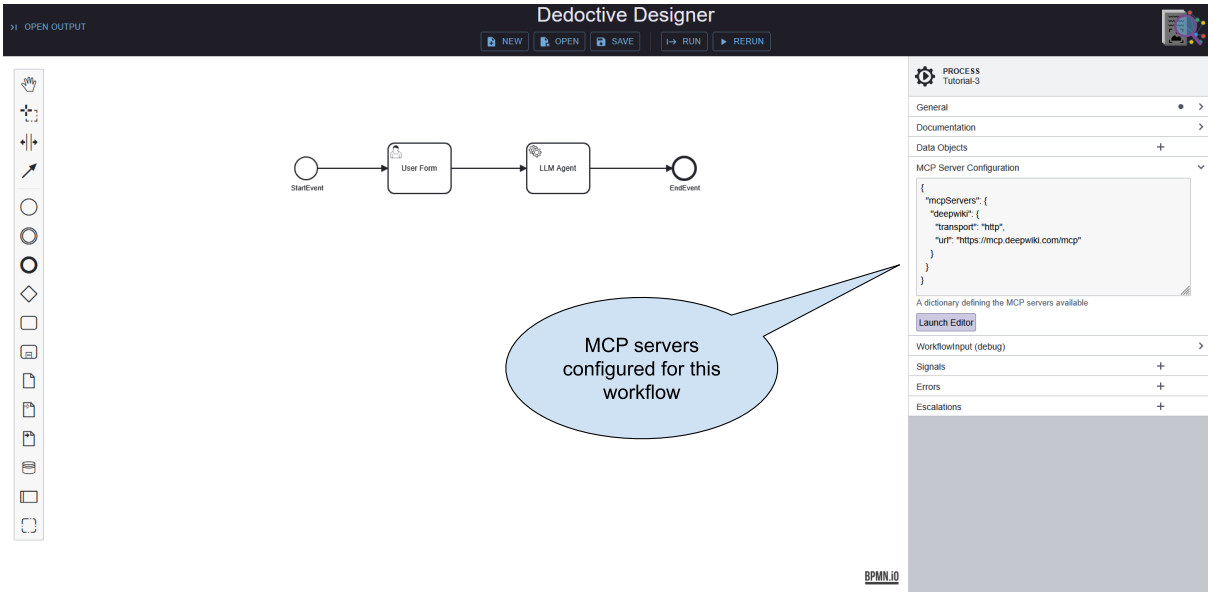
1. Starting with Tutorial-2 BPMN, delete the SimpleScript task, and add a new task in its place. Select ‘Service task’ as its type
    a. If you are creating your BPMN workflow it could be useful to rename the workflow in the Property panel under the ‘General’ section
    b. Name this task, such as ‘LLMAgent’, as this helps when viewing the Workflow Output log
2. We need to tell this workflow which MCP tools are available to it. This is done in the MCP server configuration section in the Property panel of the overall process (accessible by clicking anywhere in the background of the canvas)
    a. The format of this definition follows that of fastMCP MCPConfig
    b. Designer offers an editor, accessed from the ‘Launch Editor’ in the property panel for the overall workflow
    c. Within the MCP Server Configuration Editor enter the details of a public mcpServer such as deepwiki
        i. Server name: ***deepwiki***
        ii. Transport method: ***http***
        iii. Server URL: ***https://mcp.deepwiki.com/mcp***
        iv. Click “Add custom MCP server”



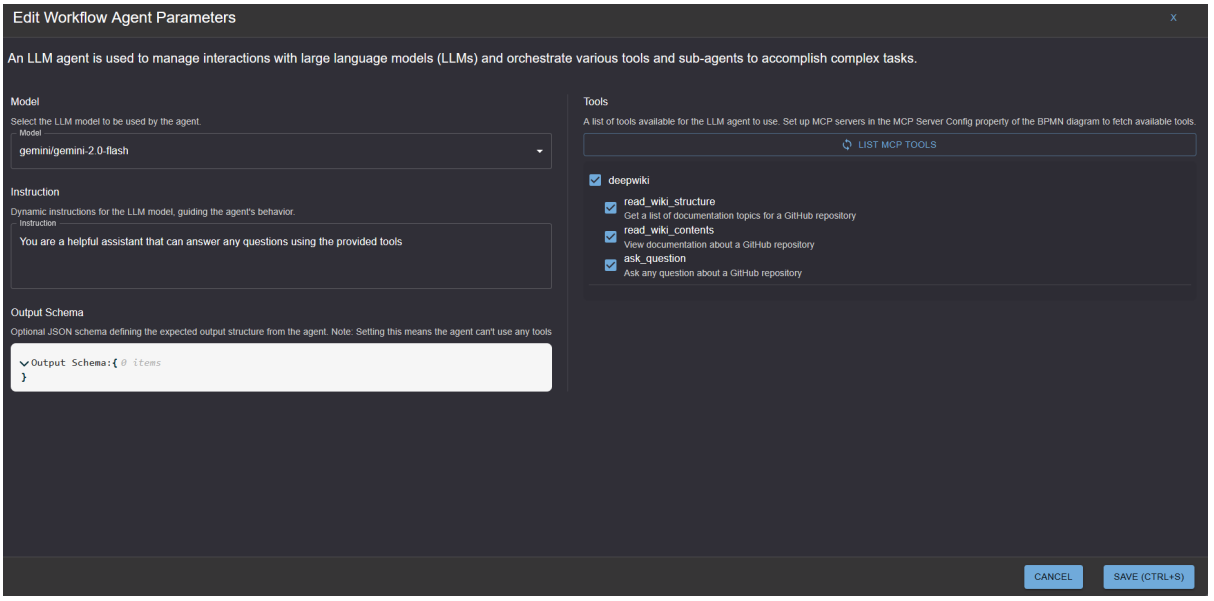
3. ***** Not included in Dedoctive Developer Edition *****
    If required this editor can specify the DedoctiveDocument MCP server that provides access to the DedoctiveDocument knowledge model. It offers tools such as list

'collections' and list 'workflows', as well as other more complex tasks for interrogating the knowledge model.

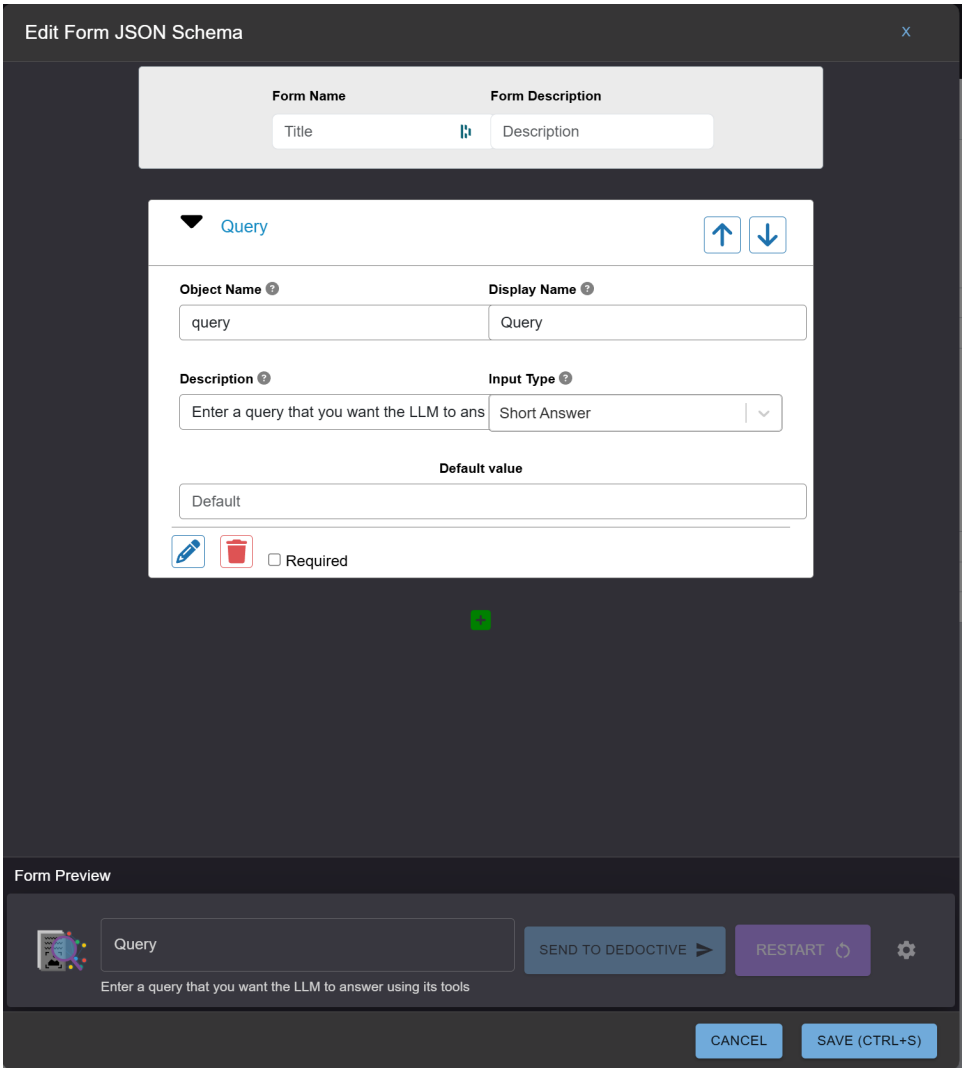
a. The DedoctiveDocument server can be added using the button.
b. To avoid explicitly defining the URL of this server, a substitution variable {{DOCUMENT_MCP_SERVER_URL}} is used instead which gets resolved by Deduction to the actual service address.



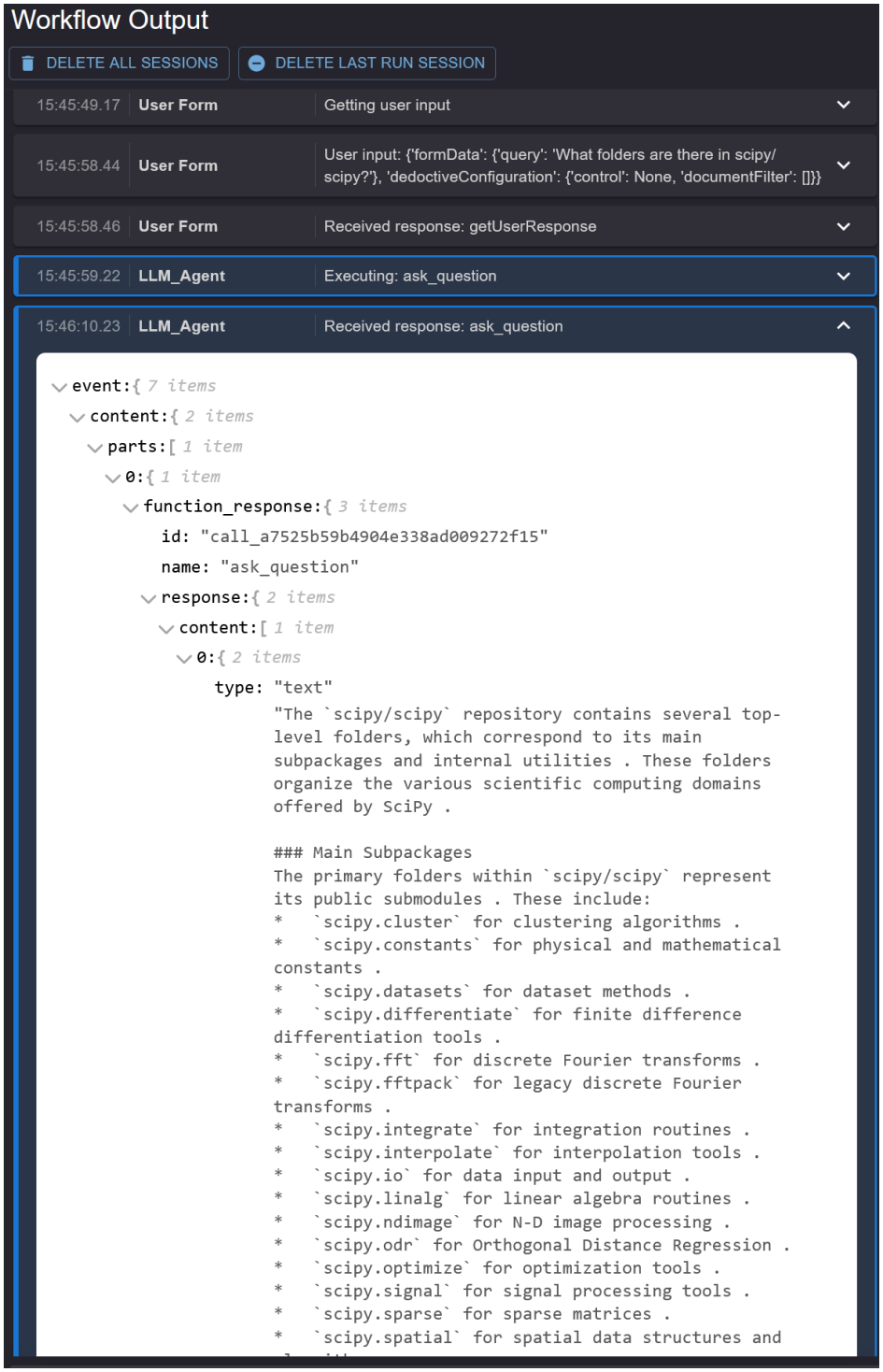
4. The next step is to configure the LLMAgent:
   a. Go to Dedoctive Service Properties in the property panel of the LLM Agent task
   b. Enter its operator ID: ***adk/LLMAgent***
   c. The remainder of the LLM Agent parameters can be defined within the Editor
      i. The model to use: ***gemini/gemini-2.0-flash*** *(or any LLMs you have provided an API key for)*
      ii. The specific instructions it should follow: ***You are a helpful assistant that can answer any questions***
      iii. Which tools it can use when answering a user's question: ***read_wiki_structure, read_wiki_contents, ask_question***
         - The list of tools will be discovered by querying the mcpServers defined in the mcpServer configuration



5. Since we want to ask the LLMAgent a textual question, we should also modify the UserForm task
    a. Change the form as follows:

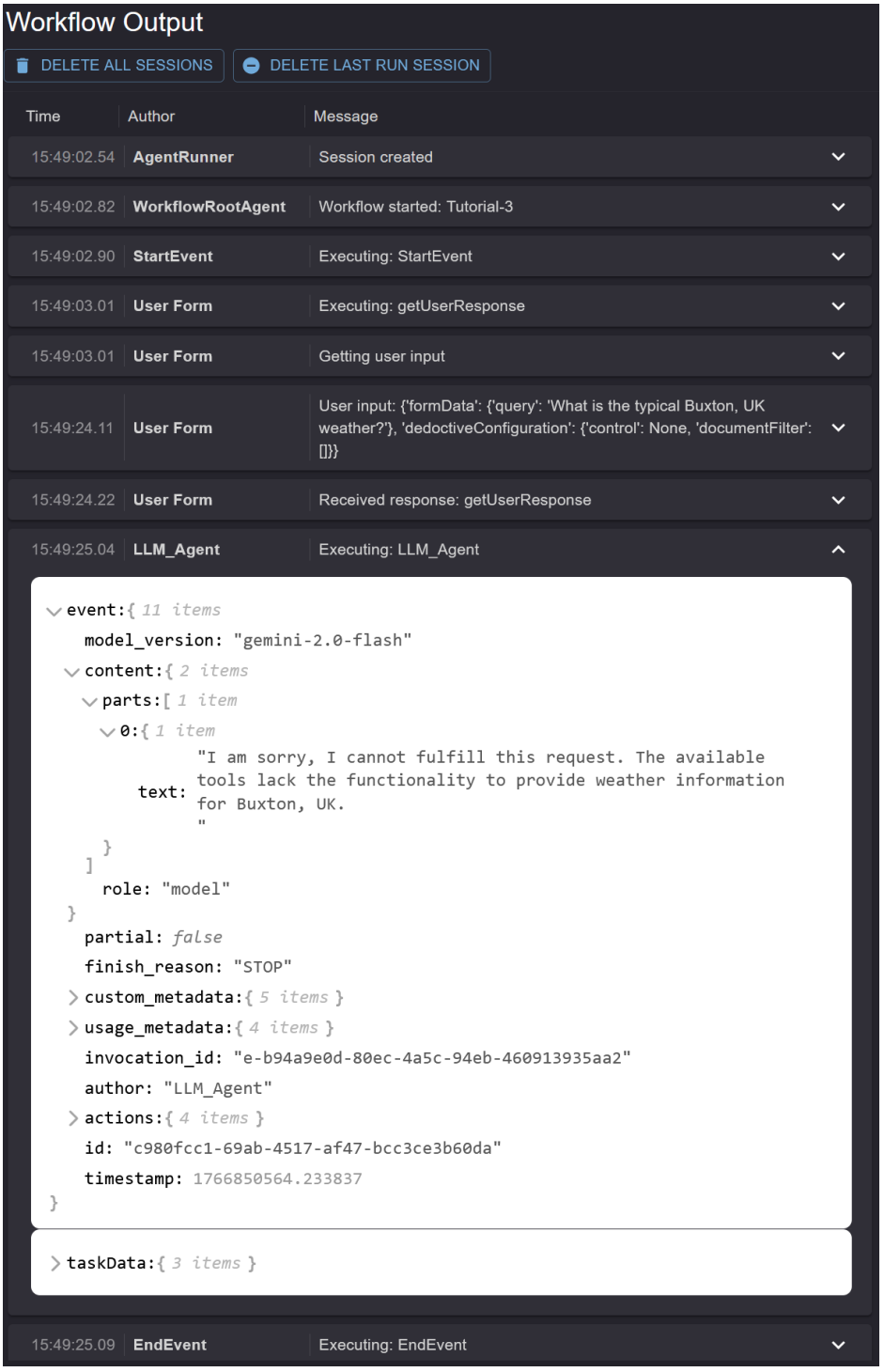


6. Run this workflow
    a. The workflow will pause at the user entry form.
    b. Enter something like '***What folders are there in scipy/scipy?***', and then 'Send to Dedoctive'
    c. In the Workflow Output panel we see that, after the UserForm 'Received response', the LLMAgent reports that it is executing *'ask_question'*
    d. After this step, there is a message *'Received response: ask_question'*, which contains the answer to our question.



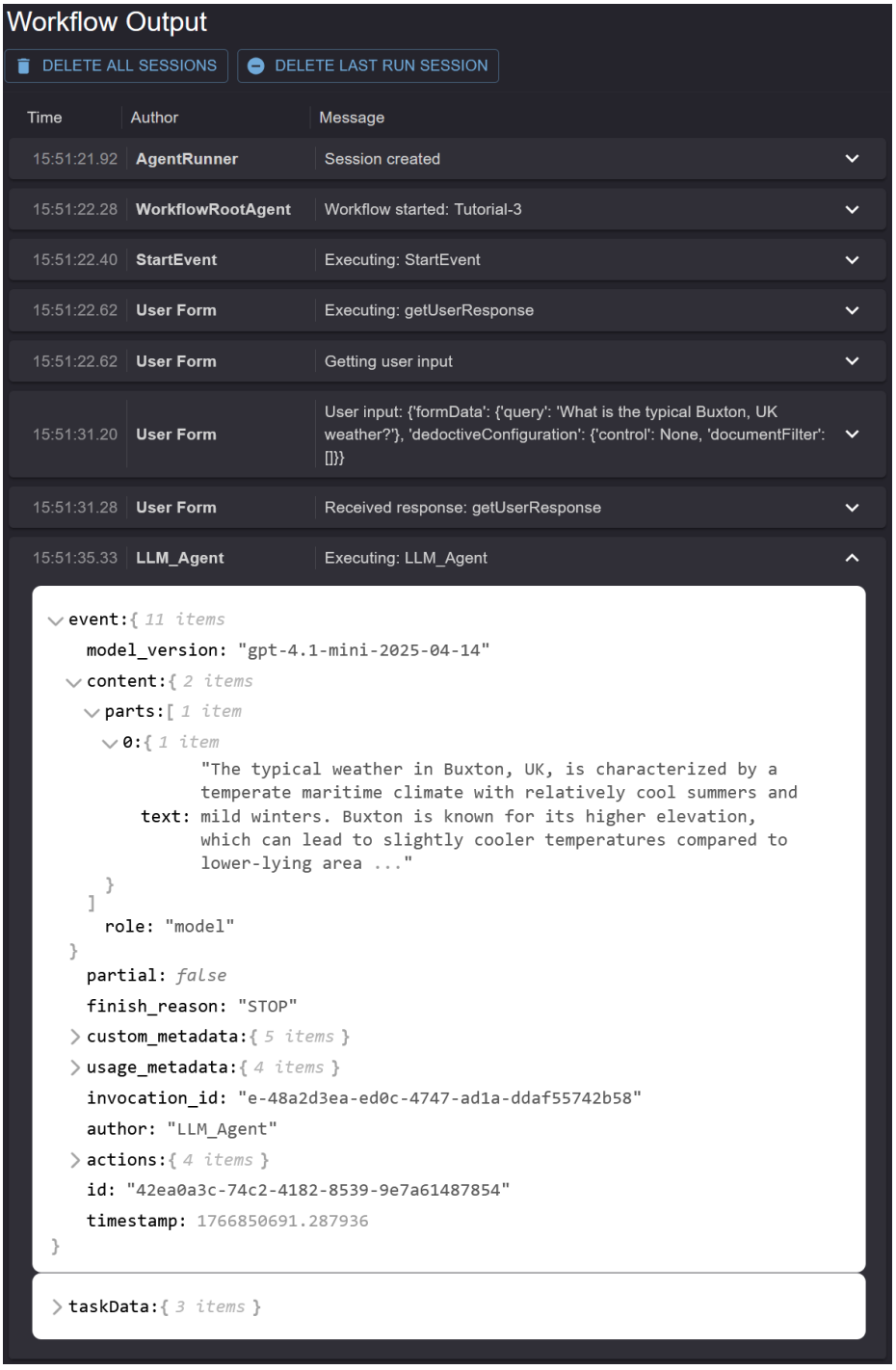
    e. Admittedly this answer is nested deeply in the event.content.parts[0].function_response.response.structuredContent.result However don't blame Dedoctive for this: we are just using the Google ADK standard which defines and supplies this dictionary structure

7. Run this workflow again with other questions
    a. For example, ask '***What is the typical Buxton, UK weather?***'



    b. The LLMAgent (with the chosen model) chooses not to answer this question. However you may experiment with other models that might be more willing to answer.
    c. For example, if you launch the LLM Agent Parameters editor you can select a different model[1]. If you can, select 'openai/gpt-4.1-mini' and rerun with the same question.
    d. This then provides this answer:

[1] The models that are available are defined in the API_KEYS environment (ENV) variable of Deduction. This variable contains model/api_key combinations. If the model required is not in the editor list, update this API_KEYS and restart deduction



3.3. Takeaways

1. A workflow can be assigned a set of tools which can be used within agentic tasks
2. These tools are all based on the MCP standard, so any MCP tool server can be used.
3. The configuration follows the ‘fastmcp’ definition MCPConfig (see https://gofastmcp.com/python-sdk/fastmcp-mcp_config)
4. Models are all accessed via LiteLLM model proxy server. This allows a wide range of models to be used. Note however that suitable API keys will probably be required. To actually use them.
5. “It is not magic, it's all behind the green curtain”. The workflow needs some prior knowledge about what the tools provided can do. This is determined on workflow startup. The workflow goes to each of the MCP servers that have been declared and

gets a full description of each tool. It is these descriptions that the LLMAgent uses to determine which ones it should call to satisfy a particular requirement

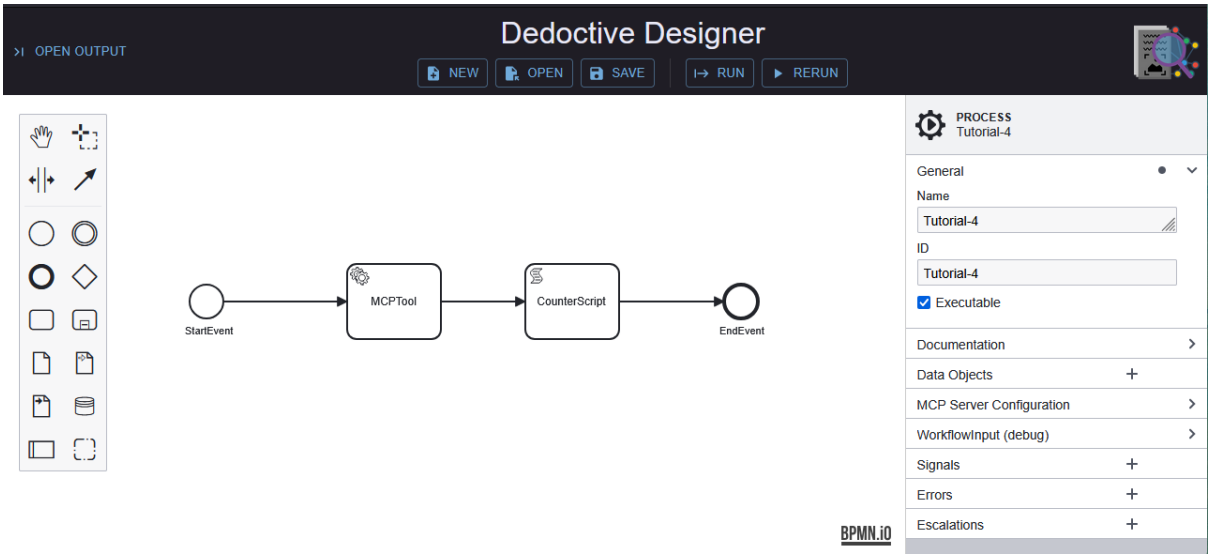
6. LLMAgents are, by definition, indeterministic. Therefore it might not answer at all. This is where Dedoctive's combination of deterministic BPMN and non-deterministic agentic workflows provides the best of both worlds. For example, following an LLMAgent, a BPMN script could examine the results and determine the subsequent workflow path.

4. Tutorial-4

4.1. Scope: Mixing BPMN-MCPTools

In the previous tutorials we have used BPMN script tasks (highly deterministic), and LLMAgent tasks (somewhat non-deterministic). In the latter case we allowed the LLM to invoke a tool if it was seen to be relevant to the question being asked. In many work-processes we will know with certainty what tool is required. Therefore this tutorial shows how a BPMN workflow can invoke a MCPTool as a task directly.

The BPMN for this tutorial can be found in Tutorials/Tutorial-4

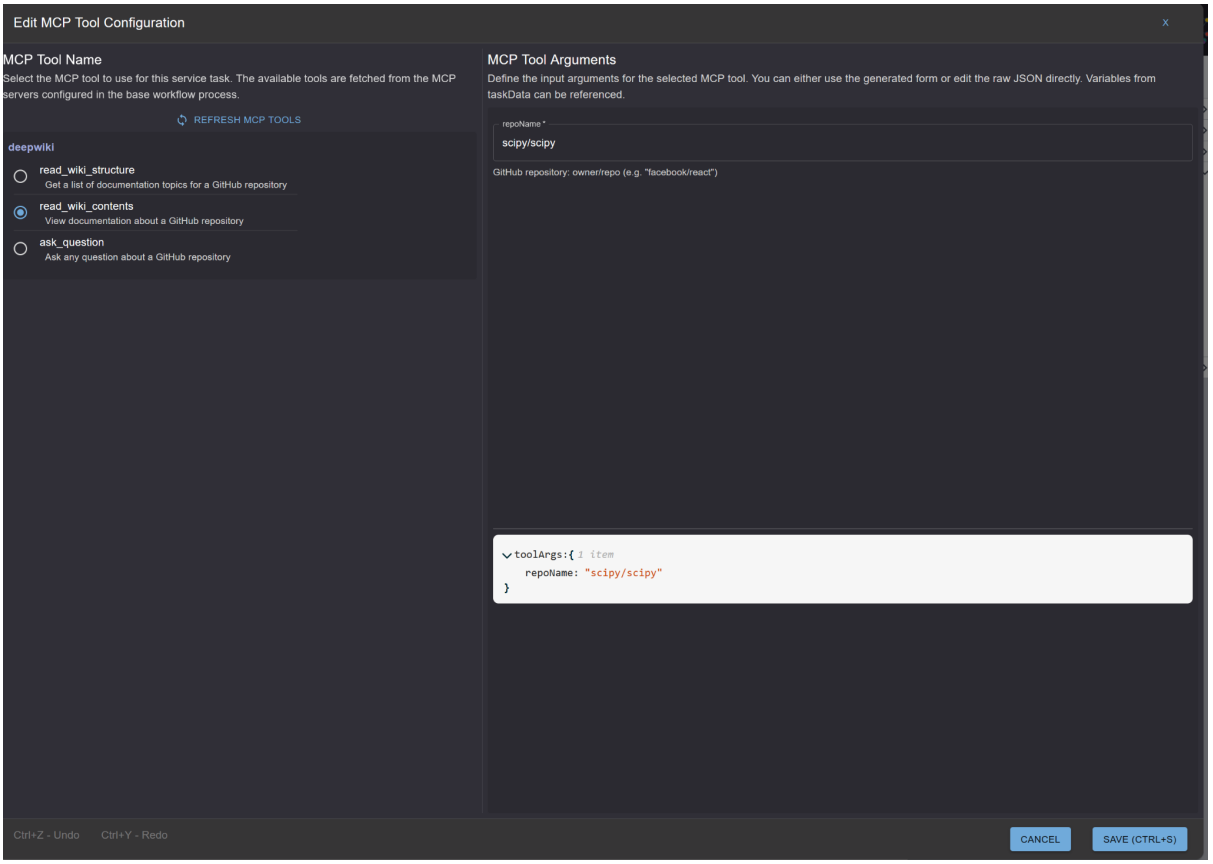


4.2. Steps

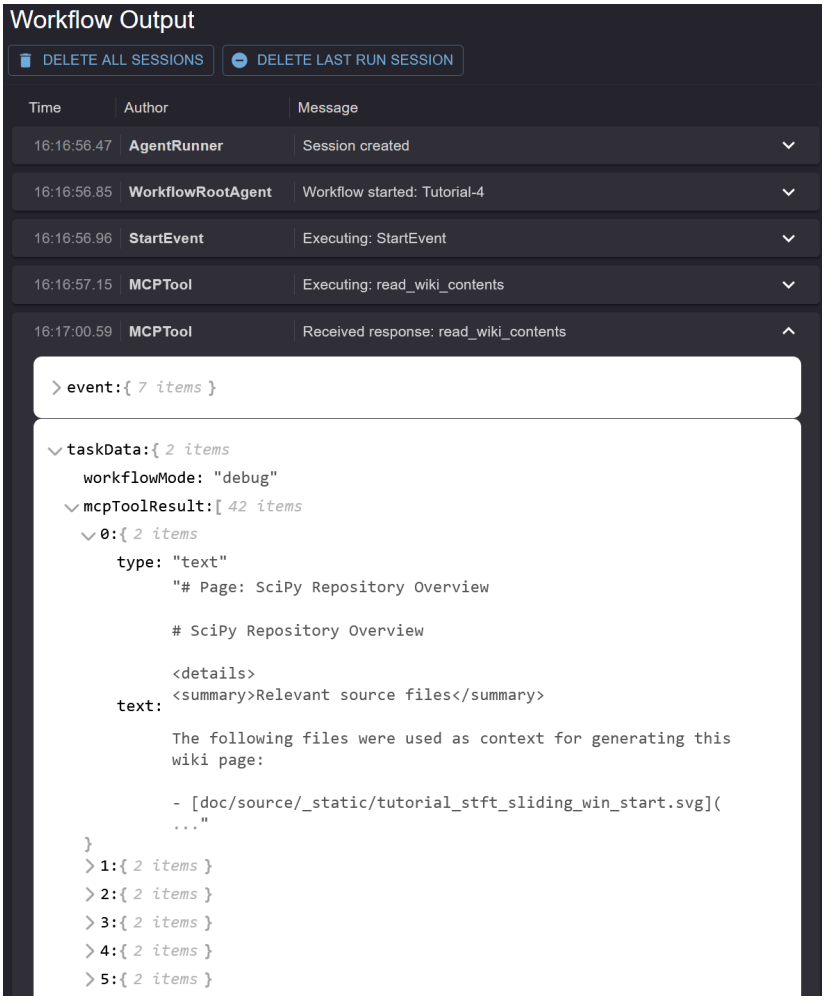
1. Either start a new BPMN workflow in DedoctiveDesigner, or start with Tutorial-3 and remove the existing tasks
    a. If starting a new workflow remember to declare a workflow name, ID, and provide a MCP server configuration
2. For this tutorial we only need a single BPMN task
    a. Set the task type as 'service task' using the task context menu spanner
    b. In the task's property panel, assign a name and ID as 'MCPTool'.
        i. The ID can be left as system assigned but using your own name is easier for debugging
3. In the service task's property panel assign the following
    a. Select '***mcp/MCPTool***' as the Operator ID
    b. Assign '***mcpToolResult***' as the response variable
        i. This is where the mcpTool will place the results of the call

ii. The results will also be in the event associated with running this task, but as we have seen, it can be nested quite deeply within the event dictionary

4. Launch the Editor in the MCP Tool Configuration
    a. Select ***read_wiki_contents*** from the list
    b. This tool requires a ‘repoName’ as an argument (discovered when the mcpServers defined at the workflow level were searched). Enter, say, ***scipy/scipy***



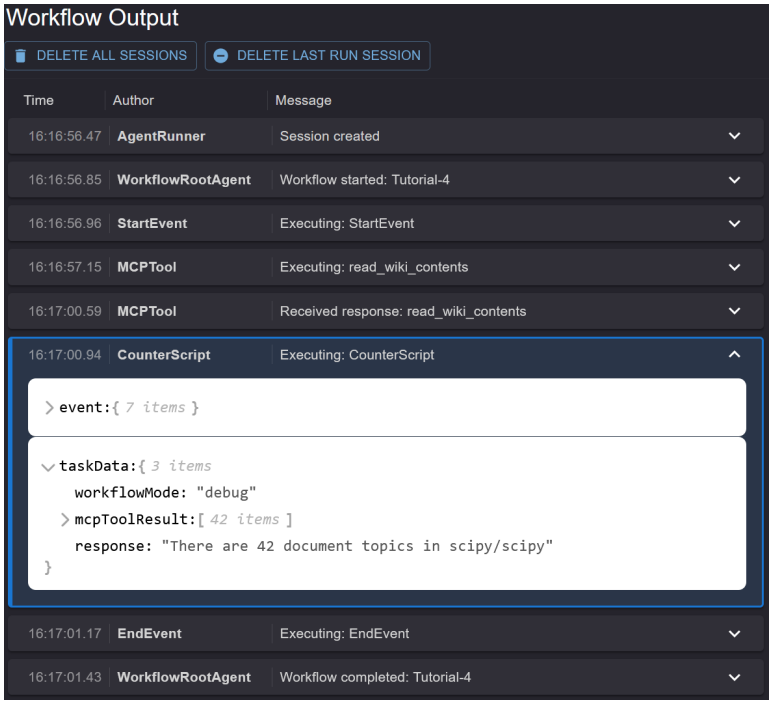
5. Run the workflow
    a. Since there are no user inputs required, the workflow runs through to completion
    b. In the Workflow Output panel we can see it executed the MCPTool and successfully received a response
    c. If we look at the EndEvent details we can see that there is a variable ‘mcpToolResult’ in the taskData dictionary.
        i. This is the variable that was defined as the Response Variable for the Service task
    d. This variable contains the structured output from the mcpTool that was called



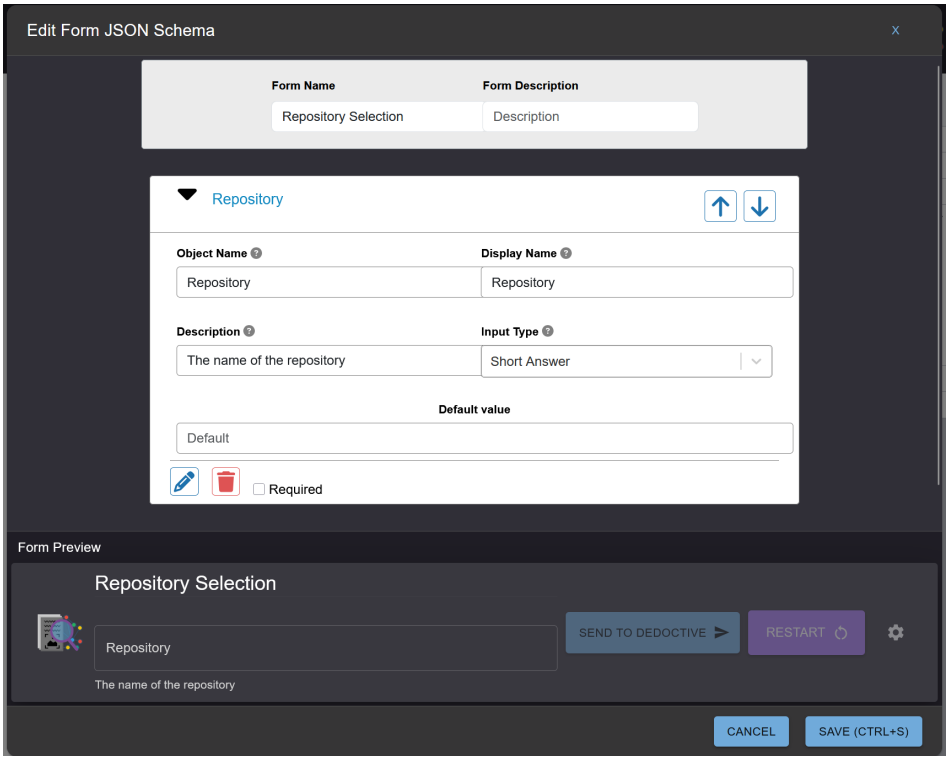
6. Add a new script task after the MCPTool Service Task
    a. Call this script task "***CounterScript***"
    b. Use this as its script "

```
response = "There are " + str(len(mcpToolResult)) + " document topics in scipy/scipy"
```

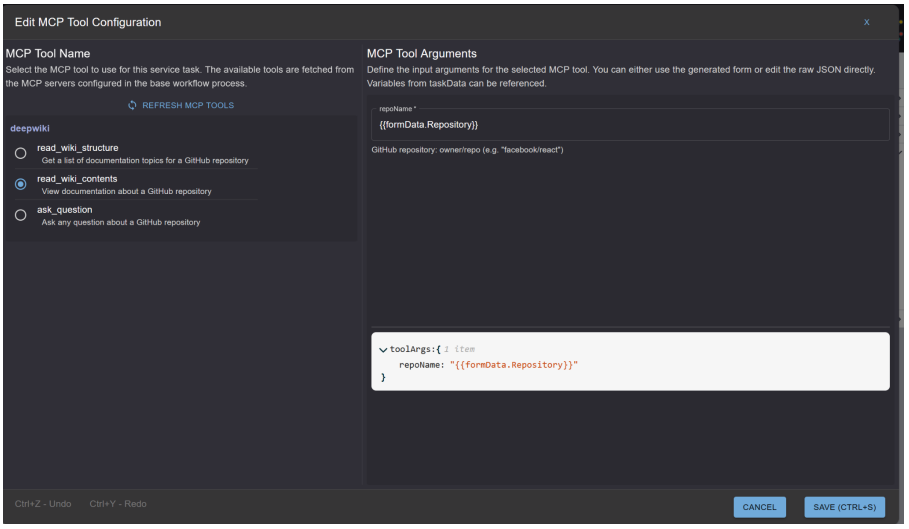
7. Run this workflow again
    a. We see in the Workflow Output panel that the CounterScript successfully executed
    b. Examining the 'taskData dictionary we see the response as expected.



8. Hardcoding the repository name is not really that useful, so we can add a UserForm, and use the value entered in the UserForm to modify the MCPTool's input arguments.
    a. This modified tutorial is saved in Tutorial-4/Tutorial-4.1.bpmn
9. Start by adding a new 'UserForm' task to the workflow prior to the MCPTool task.
    a. Configure this UserForm to prompt the user for the repository name.
    b. The UserForm configuration screen is shown below.
    c. Note that the object name is '***Repository***'
    d. This will appear within the taskData as formData.Repository

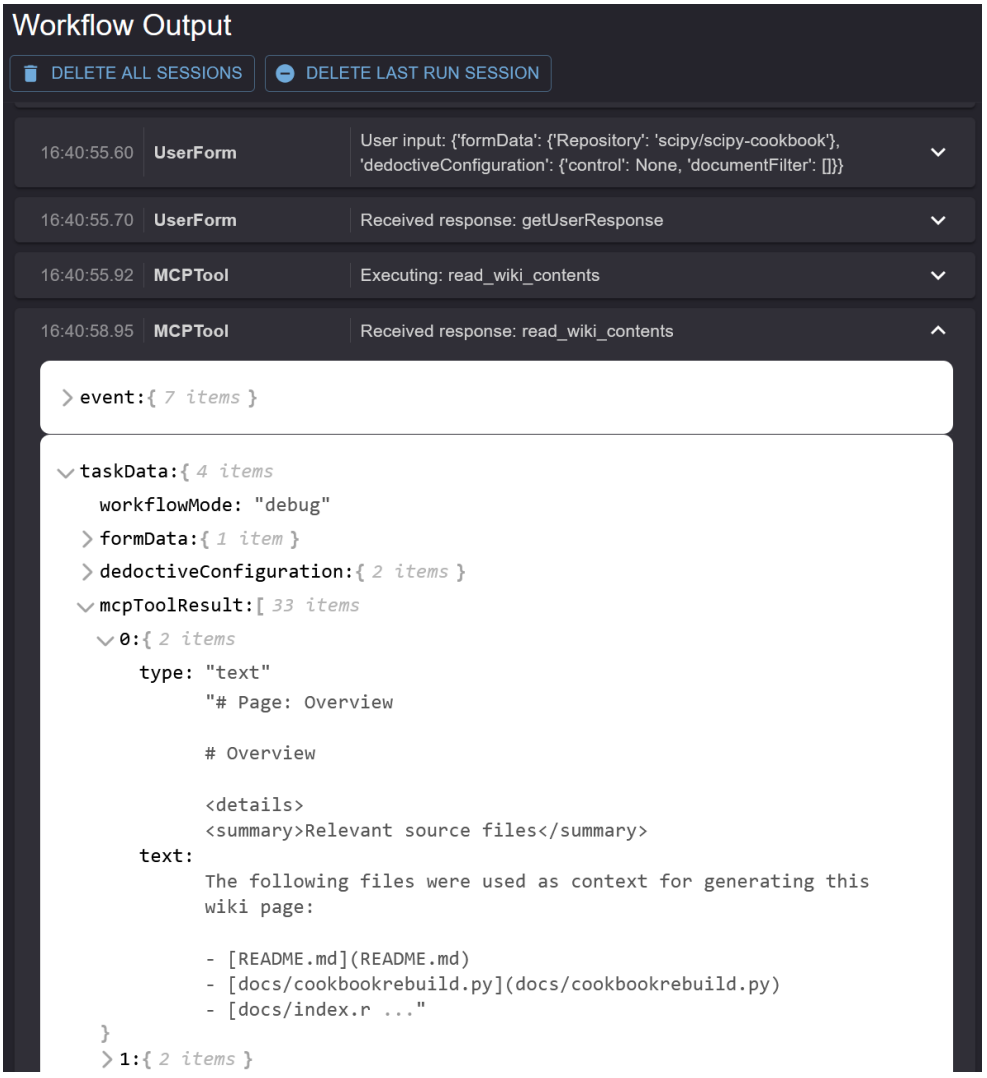


10. Next we go back to the MCP Tool task and launch the editor again
    a. Modify the tool arguments of the MCPTool task to use a reference to the formData, as shown below:



b. To reference variables in taskData use the {{...}} convention, where … can be any expression. In this case use {{formData.Repository}}, this substitutes in the value of the Repository element in the form.

11. Run the workflow again.
    a. When prompted for a Repository, add something like '***scipy/scipy-cookbook***', just to be different than before
    b. Once entered, the workflow will proceed to the MCPTool task.
    c. Before executing the MCPTool task, Deduction will assemble the mcpTool input variables from the expression provided as the tool argument.
    d. We can see in the following snippet that the mcpTool has responded correctly and stored the result in the response variable we previously provided (mcpToolResult).



4.3. Takeaways

1. If the business process is certain that it needs to run a particular mcpTool to continue, then use a mcpTool service task instead of relying on an LLMAgent invoking that tool
2. The mcpTool will assign its results to the ‘Response Variable’ within the taskData dictionary
3. In fact any of the Service tasks will assign their results to this variable. Therefore in the case of an LLMAgent it is easier to use this than hunt through the event dictionary
4. The taskData dictionary will be available to all subsequent tasks within the workflow (unless deliberately filtered out, a special case when workflow paths split and re-merge)
5. As many mcpTools will take arguments, these can be assigned in MCP editor tool argument
6. The tool arguments can be constructed from any of the data being passed forward within the workflow.

5. Tutorial-5

5.1. Scope: BPMN Workflows can be invoked by BPMN workflows

So far we have created very simple agentic and workflow processes. In more realistic use-cases these processes will be more complex involving:

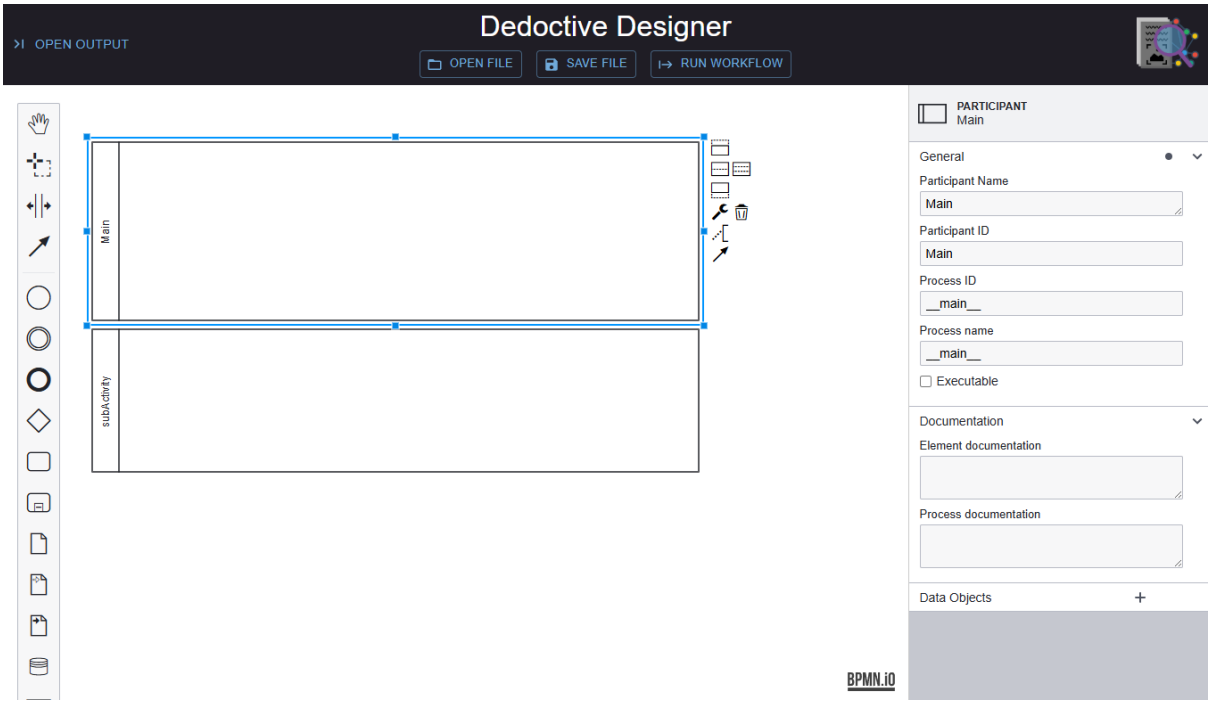
1. Conditional branching and merging of the process paths
2. Iterations and loops
3. Modularization into subprocesses and workflow

It is the latter modularization topic we want to introduce in this tutorial. This is in part as a segue to the next tutorial that will introduce agents invoking workflows, another aspect of modularization.

The BPMN for this tutorial can be found in Tutorials/Tutorial-5

5.2. Steps

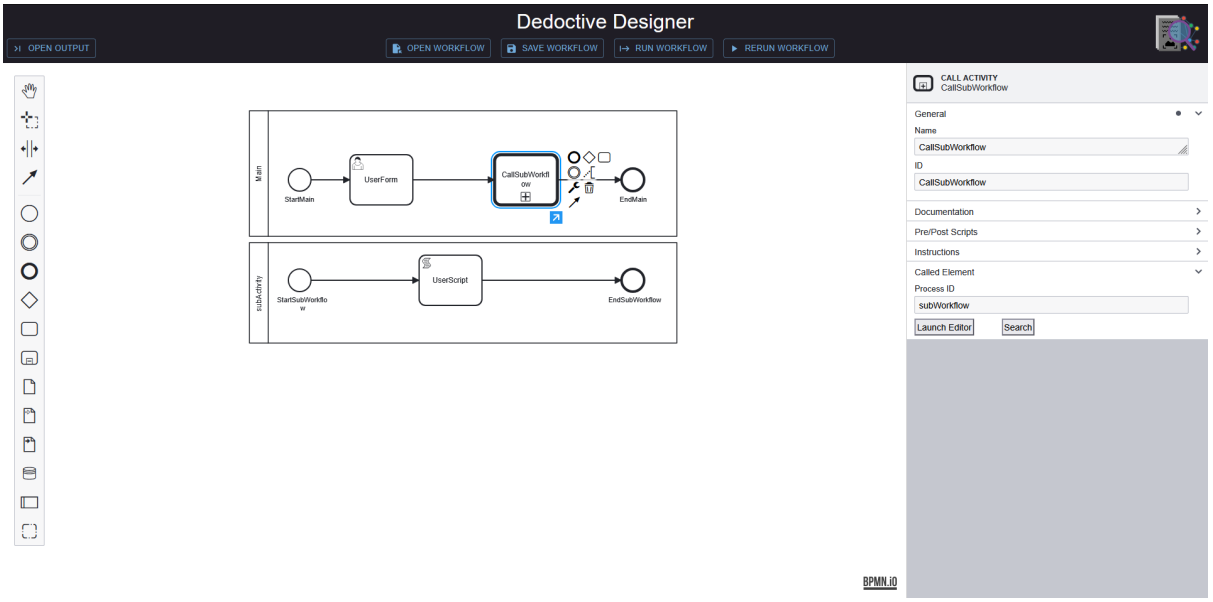
1. Create a workflow with two pools/participants
    a. We want two workflows. To show they are different workflows we put them into pool/participant containers on the same canvas as follows
    b. Starting with a blank canvas add two pools/participants (second from the bottom in the palette on the left)
        i. Note that the overall canvas is now termed a ‘Collaboration’. Without a pool/participant it would have been a ‘Process’.
        ii. Click the background of the canvas to edit the Collaboration and set the ID to ***Tutorial-5***
    c. Name one participant as follows;
        i. Participant name : ***Main***
        ii. Participant ID: ***Main*** (optional but helps debugging)
        iii. Process ID: ***__main__***
            1. obligatory since there are two workflows the Deduction engine needs to know which one to run by default
        iv. Process name: ***__main__***
    d. Name the other one as follows;
        i. Participant name : ***subActivity***
        ii. Participant ID: ***subActivity*** (optional but helps debugging)
        iii. Process ID: ***subWorkflow***
        iv. Process name: ***subWorkflow***
    e. Your canvas should now look like this



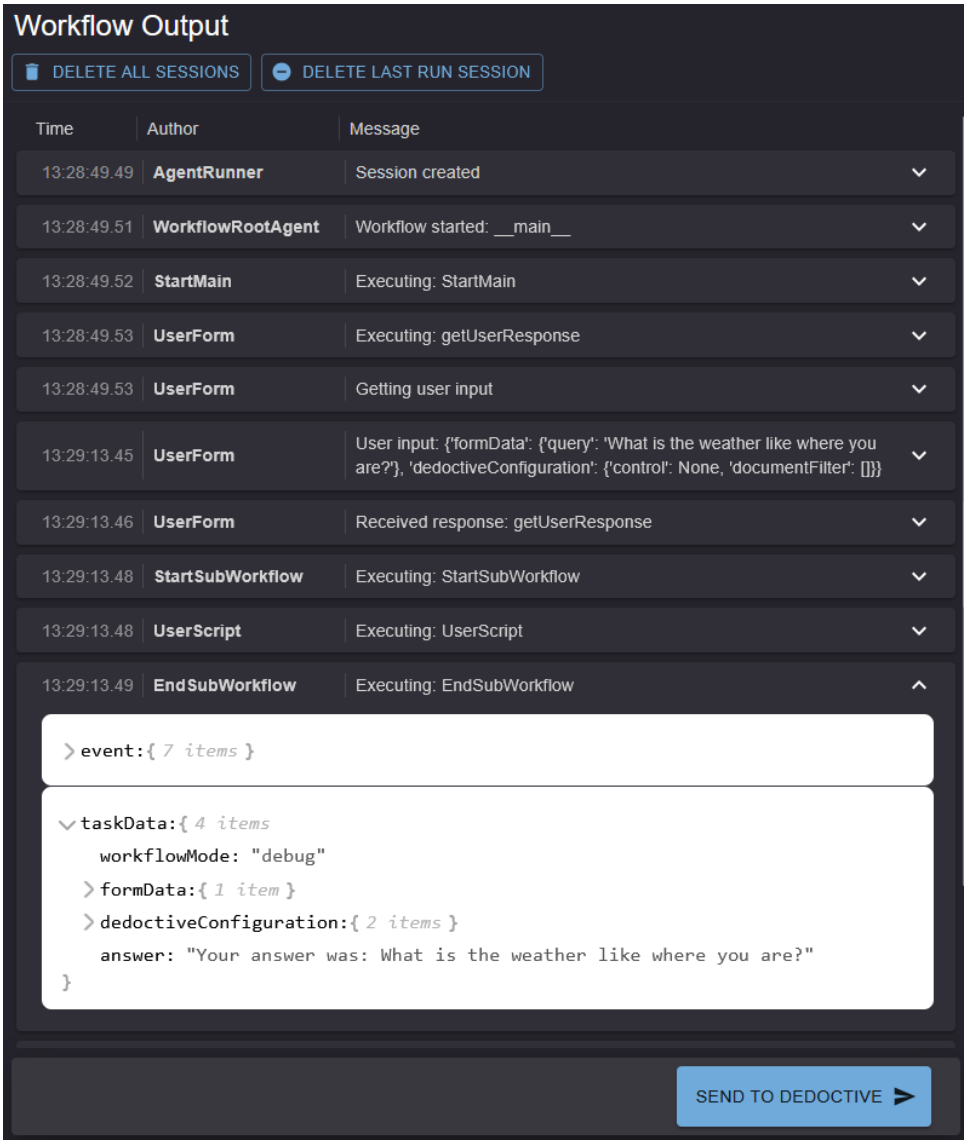
2. Populate the Main participant with a ‘Call Activity’ task
    a. Add start, and end events to the Main pool/participant
        i. Label them ***StartMain*** and ***EndMain*** respectively
    b. Add a UserForm
        i. Open the form editor, create a new form element and set object name to ***query*** and display name to ***Query***
    c. Add another task between the UserForm and EndMain
        i. Label it ***CallSubWorkflow***
    d. Select ***Call Activity*** as the task type
    e. In the Property panel of the CallSubWorkflow task, expand the ‘Called Element’ section and assign process ID to ***subWorkflow***
        i. This is the ProcessID of the workflow we want the ‘Call Activity’ to invoke.
        ii. This means we can have multiple subWorkflows on the same canvas as long as they are distinguished by their ProcessID and set up as separate participants.
3. Populate the subActivity participant
    a. Add start, and end events to the subActivity pool/participant
        i. Label them ***startSubWorkflow*** and ***EndSubWorkflow*** respectively
    b. Add a Script task called ***UserScript***
        i. The script can manipulate the resultant formData such as:

```
answer = "Your answer was: " + formData.get('query')
```

    c. Your canvas should now look like this:



4. Run the workflow
    a. Run Workflow will launch __main__
        i. If the Deduction engine cannot find the __main__ process it will report an error and stop
    b. The __main__ workflow runs, and then invokes the subActivity which continues until it encounters the UserTask
    c. As in any workflow, the process is suspended until the user responds when the workflow continues until completion.
    d. If we examine the EndSubWorkflow in the Workflow Output panel we can see both the formData and the answer generated by the script.



5.3. Takeaways

1. There are many ways available to control the process flow in a BPMN workflow:
    a. Conditional branching and merging of the process paths
    b. Iterations and loops
    c. Modularization into subprocesses and workflow
2. Modularization allows a Call Activity task to invoke another process (aka workflow) which runs just as if it were inline with the main process.
3. This provides a way of structuring complex workflows. It also allows the same process to be called at different points in another workflow process.
4. If a process is only going to be called once one can use the ‘Sub-process’ task. This will improve readability of a complex process, but the sub-process can only be accessed from its parent ‘Sub-process’ task
5. The taskData dictionary variable from the ‘main’ process is also available within the called process. Similarly, changes to the taskData dictionary made in the called process will be available within the main process.

6. Tutorial-6

6.1. Scope: BPMN Workflows Agents can be invoked as a workflow Task

In the previous tutorials we have seen how either an agentic service task (LLMAgent) or a tool service task (MCPTool) can invoke a mcpTool. The latter does this deterministically whilst the former will execute the tool determined by the request.

There will be many use-cases in which there is no mcpTool that performs the task required to satisfy the business requirements of the workflow. However, this 'task' could be created as a BPMN workflow as described in the prior tutorials.

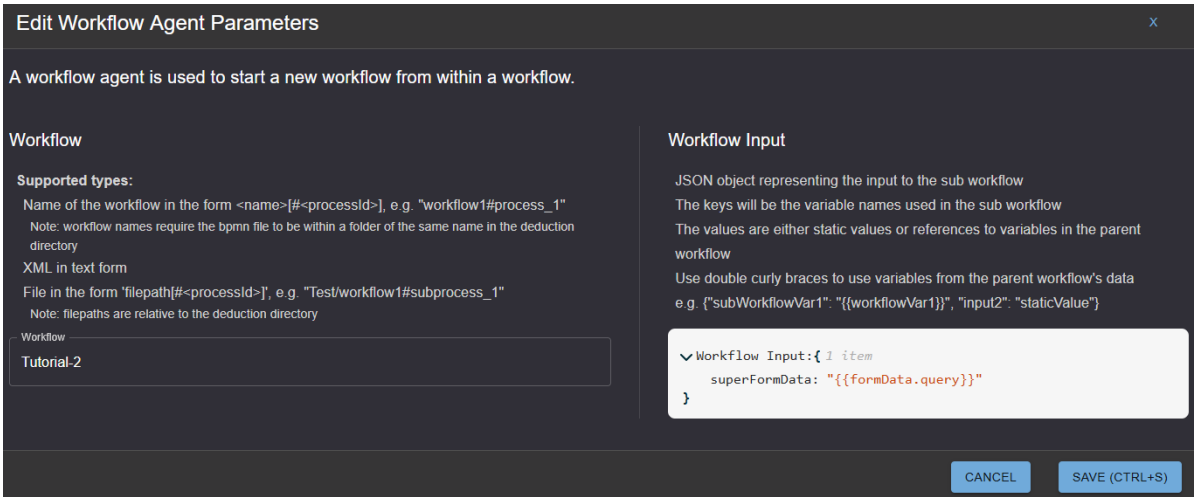
BPMN already allows one workflow to invoke another workflow, as shown in the last tutorial.

Therefore in this tutorial we show how an existing DedoctiveDeduction workflow can be used as a tool within an agentic LLMAgent

The BPMN for this tutorial can be found in Tutorials/Tutorial-6

6.2. Steps

1. We can start with the same workflow process as Tutorial-5
    a. However we do not need the subActivity participant/pool
    b. This can be left in the Tutorial-6.bpmn or removed.
2. Instead of a 'Call Activity' task as in the last tutorial, we will use a Service Task
    a. Name the task ***CallExternalWorkflow***
    b. Assign it operator of ***adk/WorkflowAgent***, because we are using an adkAgent to run this external workflow
    c. We will want to return the data generated in the called workflow to the calling workflow, so we should assign a name to the Response Variable. In this case we've used ***'workflowResponse'***
    d. We can then edit the properties of the workflow agent using its custom editor by clicking *Launch Editor*:

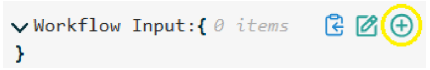


    e. The workflow is referenced either as:
        i. An identifier so that it can be found in a folder of that name in the default folder of deduction workflows *(In Developer Edition this is the Workflows folder)*

ii. A file path and name. If the path is relative, **it will be relative to the location of the workflow directory**
iii. A URL
iv. An XML string of the BPMN

f. In this case we've used the name of the BPMN file: ***Tutorial-2***
Note: this will look for a file called Tutorial-2.bpmn in a folder called Tutorial-2

g. We can assign a workflowInput expression. The result of this expression will be available to the called workflow in the 'taskData' dictionary.
i. Click the +

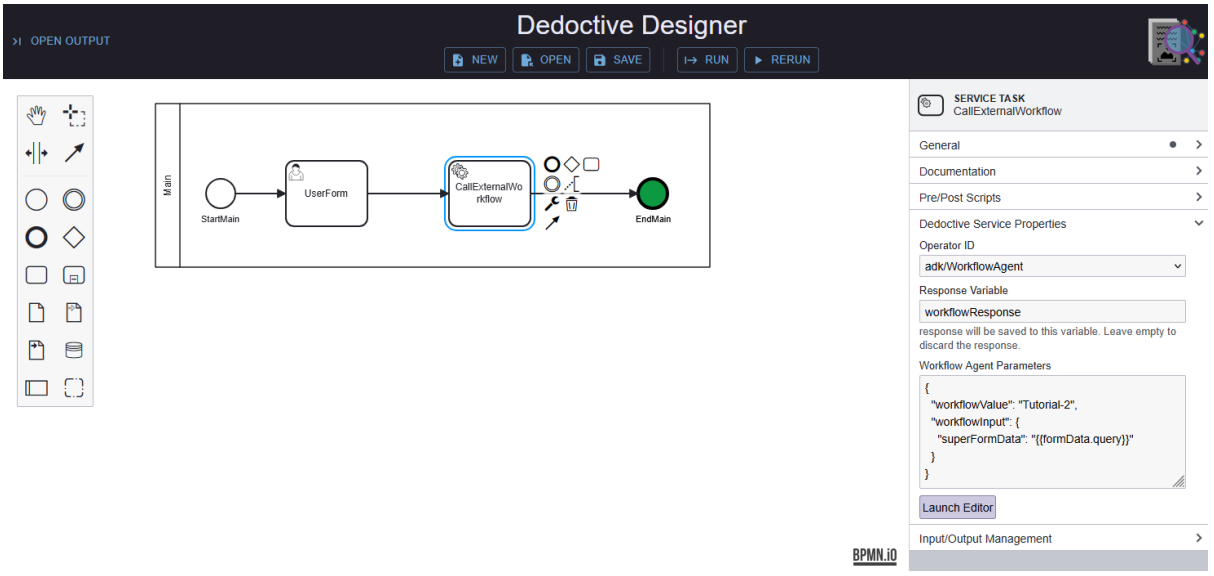


ii. Enter the key as ***superFormData***
iii. Click the edit button next to NULL and select "string" in the dropdown

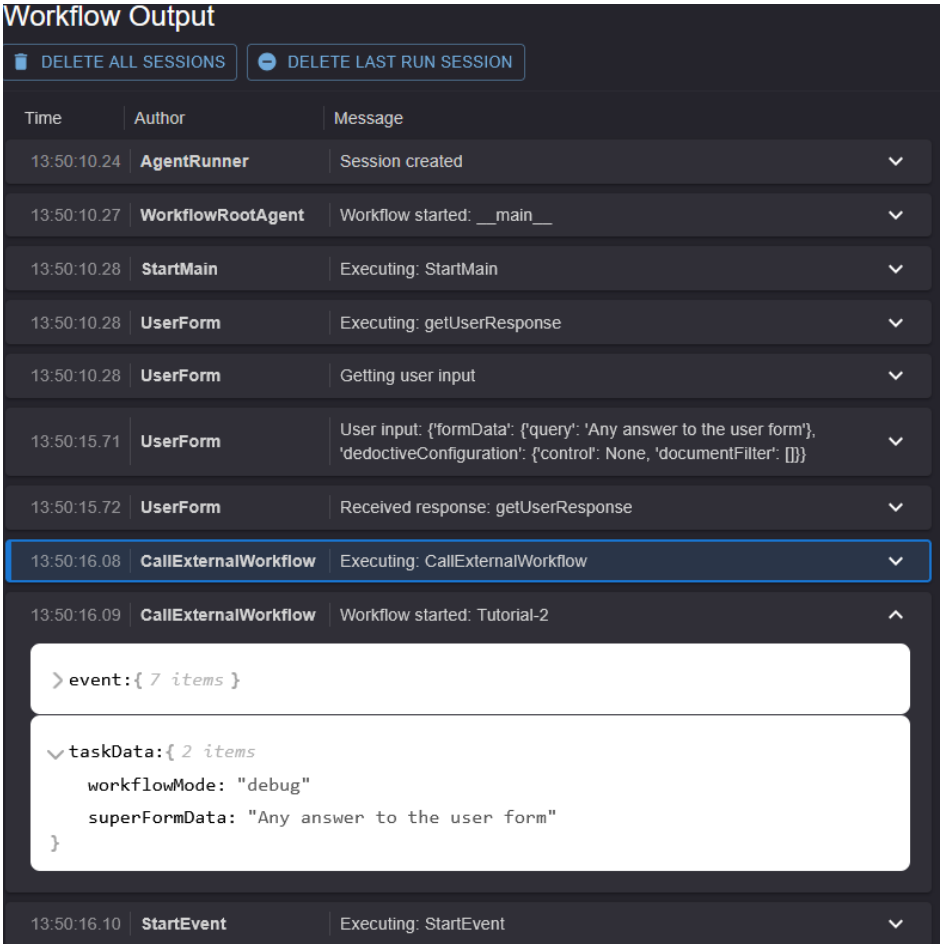
iv. Replace "null" with: ***{{formData.query}}***

h. Within this expression the contents of {{..}} will be evaluated

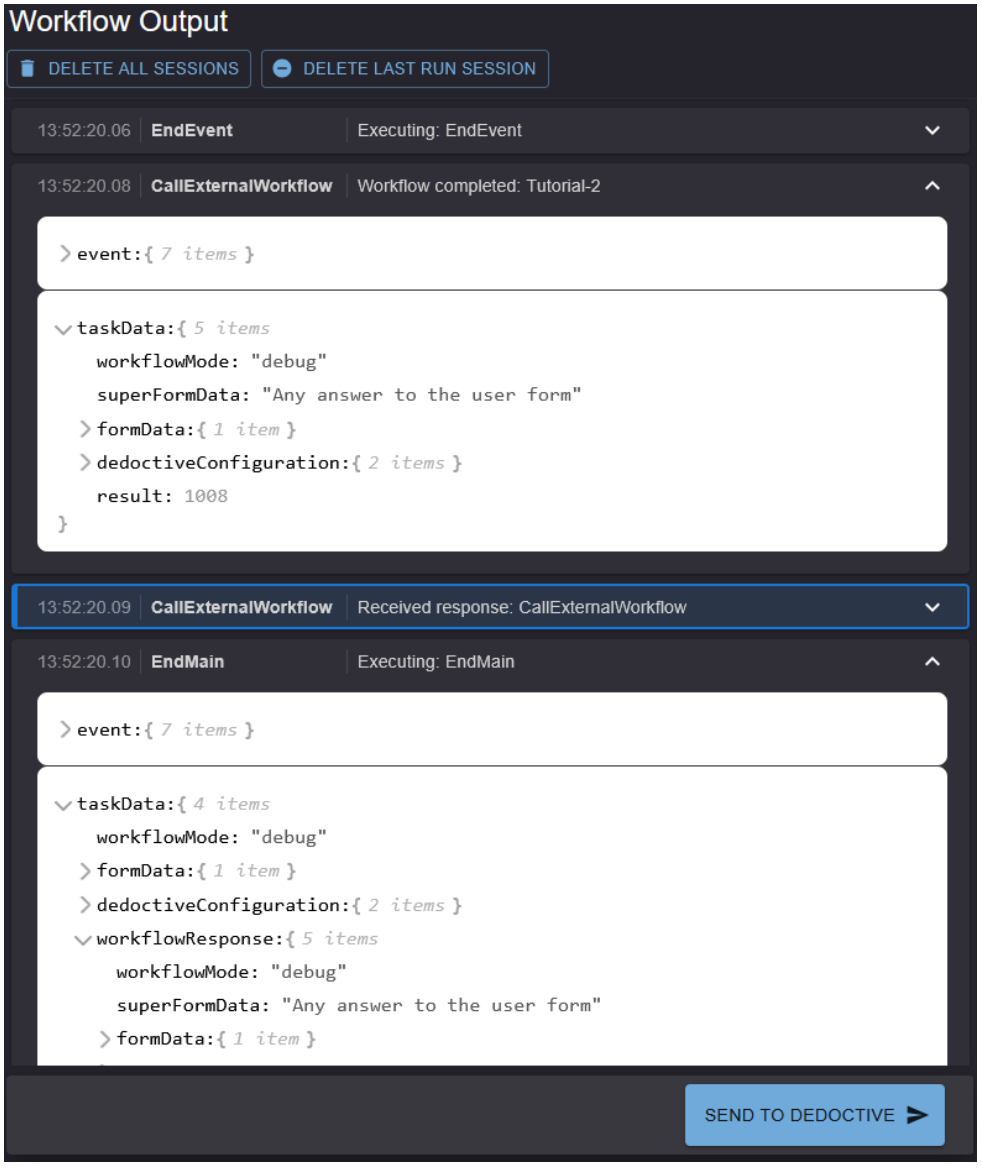
3. The canvas should now look like this:



4. Run the workflow
a. The workflow will run until the UserForm in the Main workflow, where you can enter anything.
b. The workflow will continue until it encounters the CallExternalWorkflow task.
c. It immediately loads this external workflow, prepares any input values (as defined by workflowInput), and starts executing the external workflow, in this case Tutorial-2
d. The external workflow halts for user input
e. On entry the external workflow continues until completion, when the responseVariable is assembled and returned to the calling workflow
f. The calling workflow, in this case has no more tasks so continues to completion.
g. If we examine the "Workflow started" event by CallExternalWorkflow in the Workflow Output panel we can see the 'superFormData' key in the taskData dictionary. This is what we told the service task to prepare as workflowInput



h. Further down, the “Workflow completed” event by CallExternalWorkflow shows the state of ‘taskData’ in this external workflow on completion
i. At the EndMain event of this external workflow, we can see the workflowResponse has been added to the taskData dictionary
j. The workflow then continues to completion



6.3. Takeaways

1. A workflow can be used as a tool to be invoked as a service
2. This workflow can either be named in the ‘standard’ directory of workflows, given a file path and name (relative to the DedoctiveDeduction server), referenced via URL, or as a BPMN XML string (yes really!)
3. The called workflow can be provided with workflowInput data, which is assembled from the current taskData
4. The workflow can have a response variable (just like any service task). This will be what is returned to the calling workflow.
5. The workflow-as-an-agent service task can contain any tasks, including other service tasks, agentic tasks, or even other workflows.
6. This allows workflows to be assembled as ‘tools’ that can be used by other workflows or, as they are instances of adkAgents, as any other agent.

7. Tutorial-7

7.1. Scope: BPMN Workflows as MCPTools for LLMAgents

In the Tutorial-3 we have seen how an agentic service task (LLMAgent) can invoke a mcpTool. However, we needed to add a userForm at the beginning of the workflow to request from the user what actions they want to take. The subsequent LLMAgent then handled this request. If we want subsequent actions to be taken on the results of the LLMAgent response, then these tasks can be added (deterministically) to the workflow.

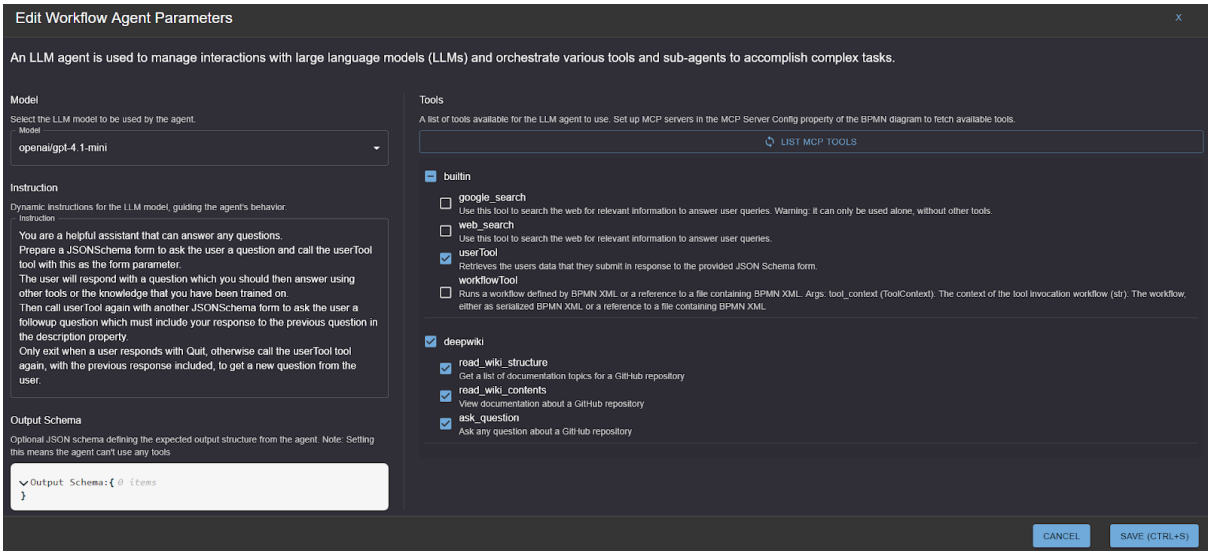
Suppose we want the LLM to handle all of this logic non-deterministically? For example, why not let the LLM ask the question and even handle the answer to perform subsequent tasks without defining them deterministically in the workflow?

BPMN already allows one workflow to invoke another workflow, as shown in the last tutorial.

Therefore in this tutorial we show how an existing DedoctiveDeduction LLMAgent can create a non-deterministic workflow process.
The BPMN for this tutorial can be found in Tutorials/Tutorial-7

7.2. Steps

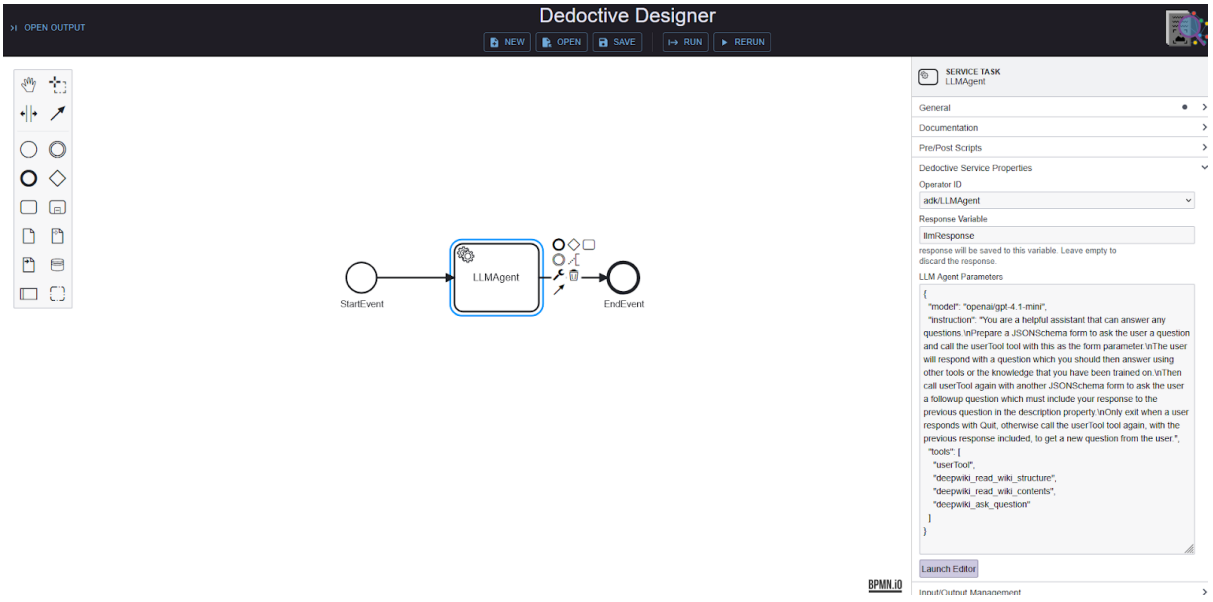
1. Create a blank workflow called ***Tutorial-7***
2. We need to tell this workflow which MCP tools are available to it. This is done in the MCP server configuration section in the Property panel of the overall process
    a. Use the same MCP Server Configuration as in Tutorial-3:
        i. Server name: ***deepwiki***
        ii. Transport method: ***http***
        iii. Server URL: ***https://mcp.deepwiki.com/mcp***
3. We only need a single Service task for this workflow
    a. Set the name to ***LLMAgent***
    b. Under Dedoctive Service Properties set the operator to ***adk/LLMAgent***
    c. The response variable is ***llmResponse***
    d. The remainder of the properties can be entered via the LLM Agent Parameters editor, or by directly modifying the JSON:



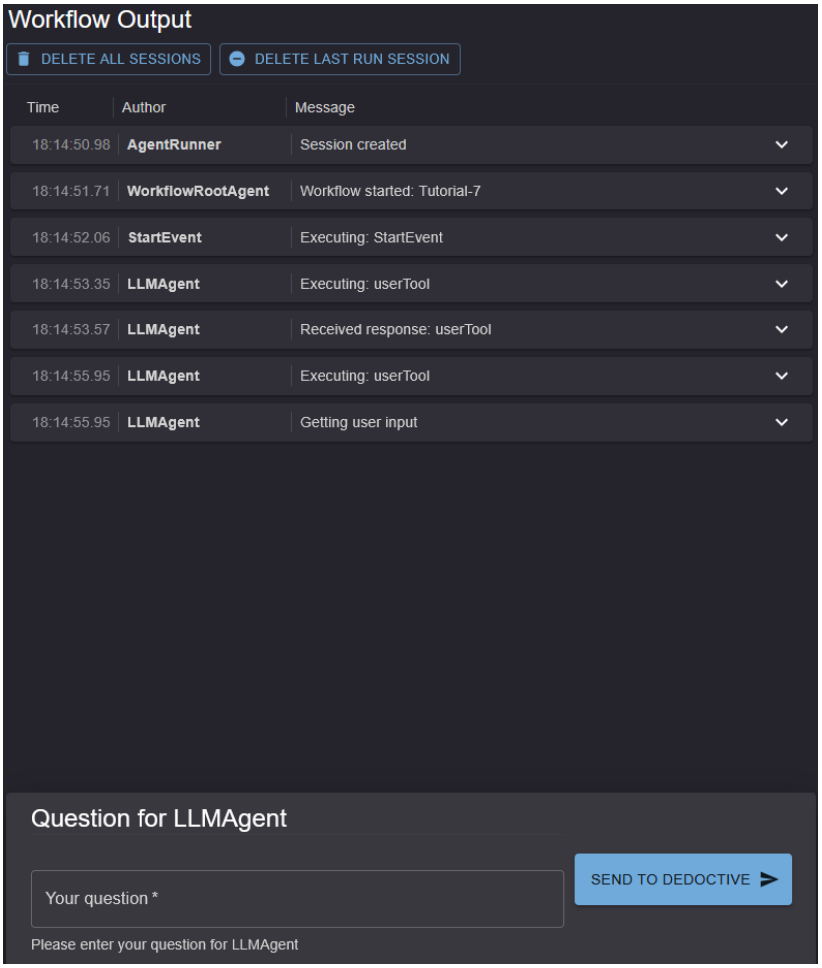
    e. The model in this case is ***openai/gpt-4.1-mini***

i. The gemini models do not seem to be smart enough for this task
ii. Other models might well rise to the challenge!
iii. If you don't have an API key for OpenAI, you can try using other models

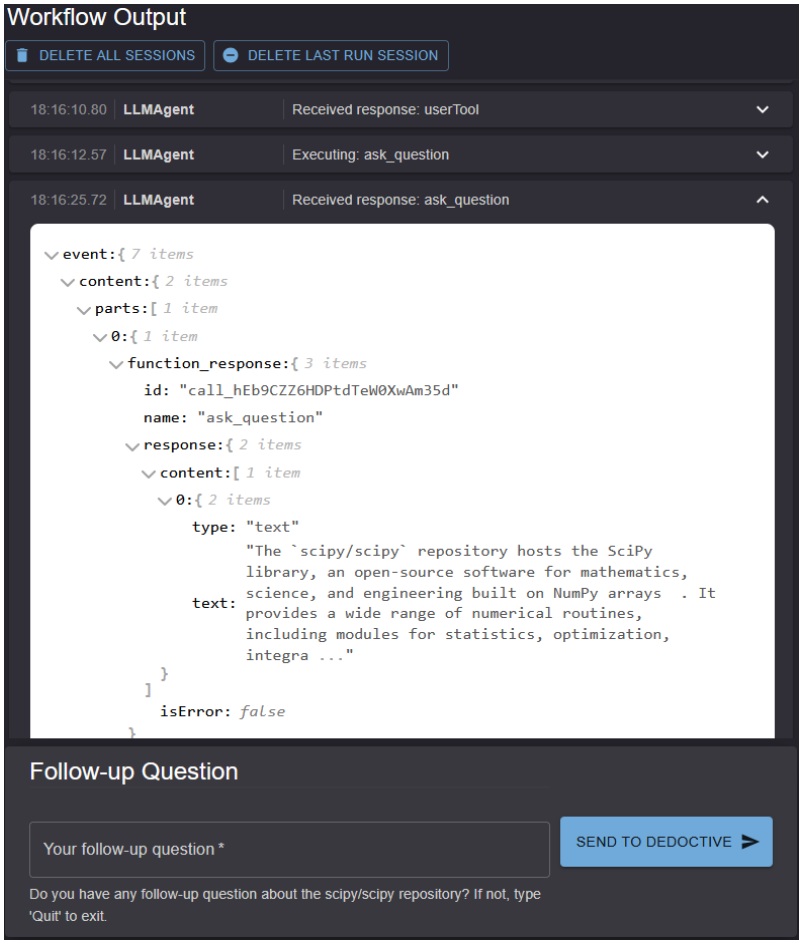
f. Add instructions
   i. ***"You are a helpful assistant that can answer any questions.***
   Prepare a JSONSchema form to ask the user a question and call the userTool tool with this as the form parameter.
   The user will respond with a question which you should then answer using other tools or the knowledge that you have been trained on.
   Then call userTool again with another JSONSchema form to ask the user a followup question which must include your response to the previous question in the description property.
   Only exit when a user responds with Quit, otherwise call the userTool tool again, with the previous response included, to get a new question from the user."
   ii. These are the instructions for the LLM, in particular how to handle the tools that are available.
   iii. We want the LLM to be in control of the discussion with the user, so we start by explaining how it can use the userTool to request answers from the user.
      1. Note that the userTool needs a JSONSchema form definition. This is something that gpt-4.1 can generate .
   iv. We then want the LLM to answer the question that it asked of the user.
   v. After that we want the LLM to recurse, by asking followup questions.
g. The tools that we want to use are:
   i. ***userTool***, a default tool that allows the LLM ask the user any question it thinks of by producing a form that is delivered to the user
   ii. ***read_wiki_structure***, ***read_wiki_contents***, and ***ask_question***
h. This configuration can be seen below:



4. Run the workflow
   a. Launch the workflow, and we should see the userForm that is generated by the LLM appear in the Workflow Output panel.
      i. Note that we now have a non-deterministic process orchestrated by the chosen LLM, so the response might not appear exactly as shown below.

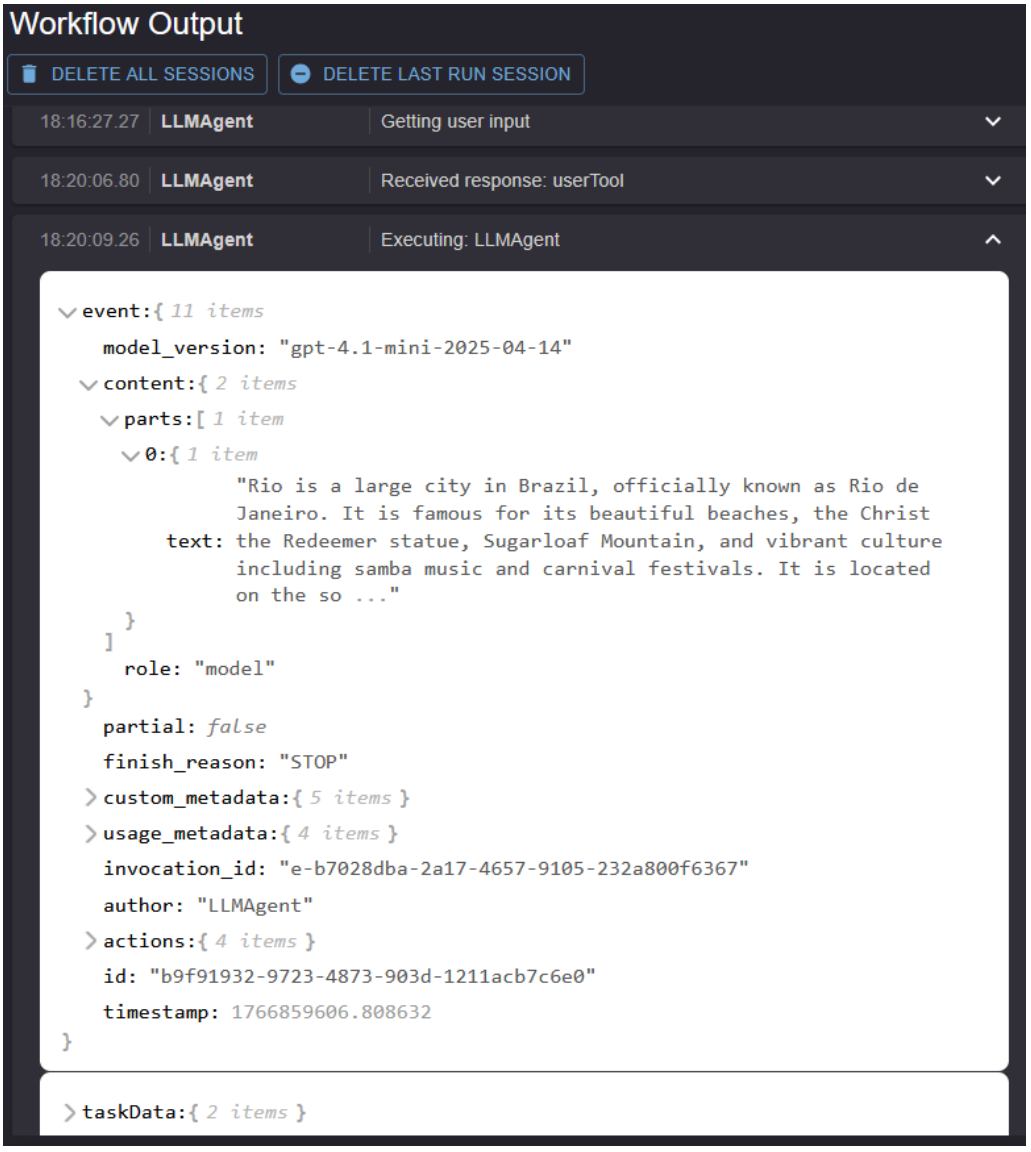


b. One can now ask any question, such as '*Please describe repository scipy/scipy*'
c. The LLM will in response to this question invoke the ask_question tool
d. It then creates another user form using the userTool as shown below.
    i. Conveniently it also displays the workflows that it found



ii. However we can also browse the events to see the actual response from when the LLM invoked the ask_question tool:

e. We can use this form to ask another question, such as ‘Where is Rio?’
   i. In other words, with just a single task we have created a full chat, together with the ability to invoke other tools
   ii. The answer to the question will appear in the events as shown below

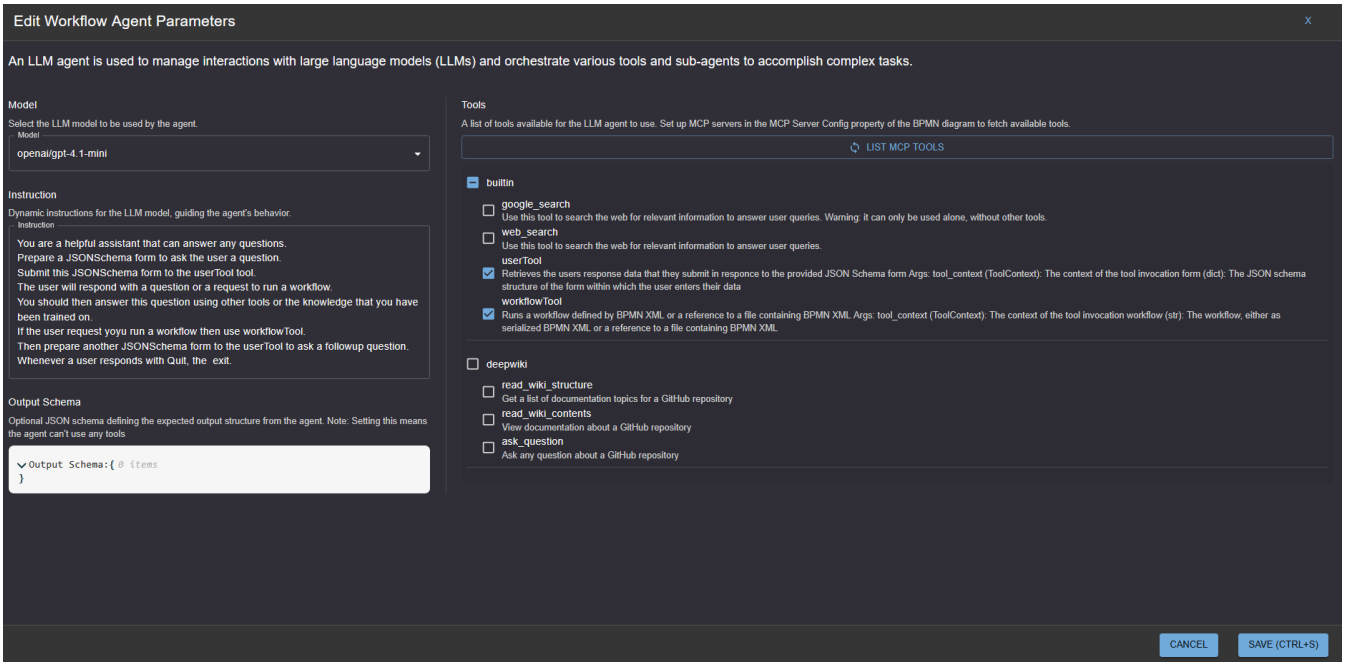


f. The same LLMAgent continues with another user prompt, following the instructions we set.
   i. This can continue ad-infinitum. In other words we have created a chat-in-a-box
g. How do we stop? Although the LLMAgent will not be deterministic with regards to handling the process flow, we find that 'Quit' causes the LLMAgent to complete.
   i. We could make this more explicit by including the stop command in the instructions.

5. Adding a workflowTool to the LLMAgent
   a. So far the tools to which the LLMAgent has access provide information back to the LLMAgent
   b. Another tool available for Dedoctive Deduction is the **workflowTool**. This runs a predefined workflow that can, of course, perform any task defined by its BPMN[2].

[2] Note that it is not the LLM that actually runs the workflow, it scarcely understands what a workflow is. Instead we are telling the LLM that there are tools available for it to call upon in the event that it cannot answer the question solely from its training. We describe what these tools do. So if we assert to the LLM: 'please run workflow x' we assume (hope) that the LLM deduces it should *ask* the caller to run the workflowTool to run 'x', and then resubmit the results of the workflowTool so that the LLM can continue. What this means is the Deduction will receive a response from the LLM requesting that Deduction runs workflow x and return the results to the LLM.

c. We can add this tool to those available to the LLMAgent service task as shown below:



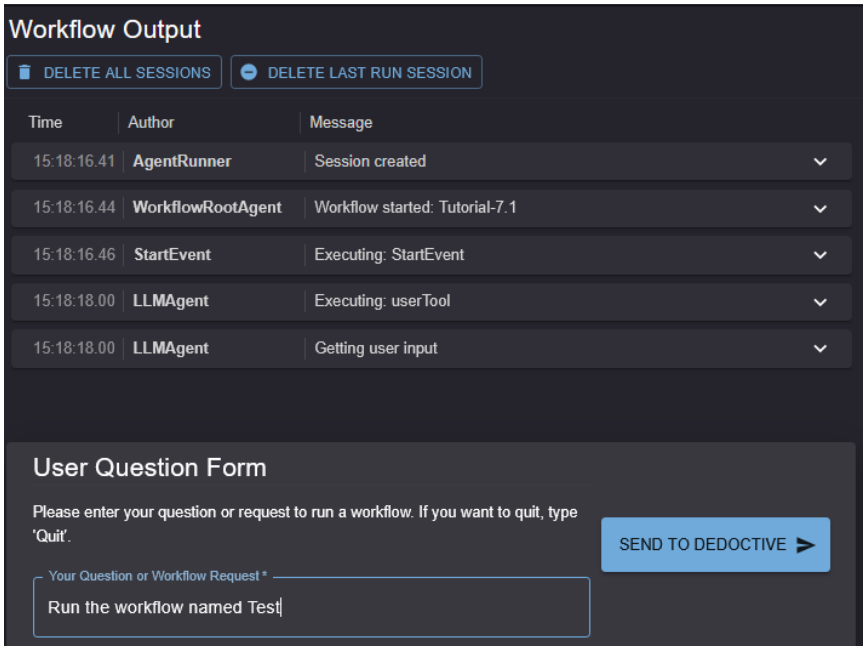
d. Note that the instructions have been extended to help the LLM interpret requests to run other workflows.

e. This modified workflow is available in Tutorial-7.1

6. Run the modified workflow

a. Launch the workflow

b. As before the Workflow Output will show a form (generated by the LLM) that asks a question



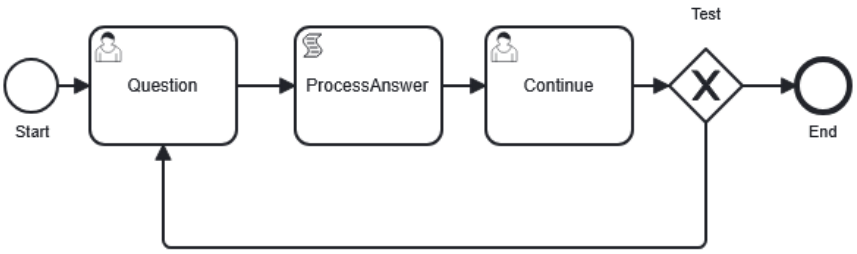
c. The response will be ‘Run the workflow named Test’

i. This is not really asking a question but issuing an instruction to the LLMAgent

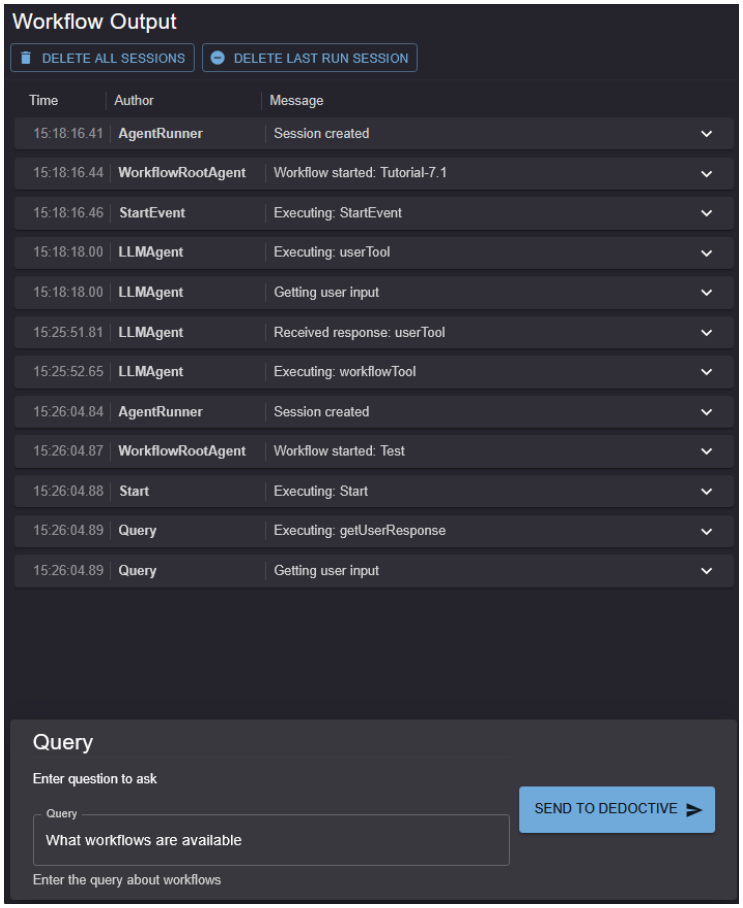
ii. The ‘Test’ workflow is a simple workflow that asks a question, processes the answer, and then checks if there is going to be another question:

iii. Deduction will try and discover this workflow definition:
   1. If it is a name, Deduction will look for this workflow in the default folder where workflows are stored.
   2. If the LLM has generated BPMN XML, Deduction will validate the correctness of the BPMN and try to run it.
   3. If it is a URL, Deduction will navigate to that URL and retrieve the BPMN XML.
   4. If it is an absolute file path, Deduction will go to that path to locate the BPMN XML.

iv. In this example, case 1 above, Deduction will attempt to retrieve Test BPMN XML from the default folder:

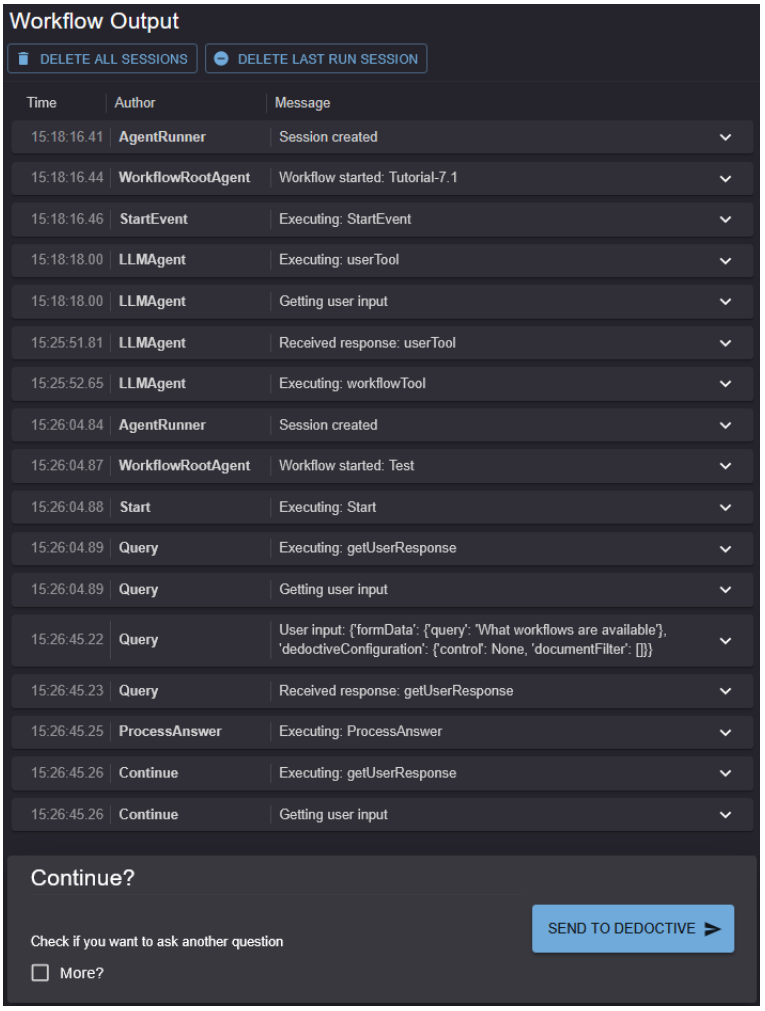


d. The LLMAgent then uses the workflowTool to launch this Test workflow.
   i. The progress of this invoked workflow is also shown in the Workflow Output panel



ii. This shows a userForm. However this is not an ad-hoc form created by the LLM, but the userForm defined in the Test workflow deterministically.. The default question is ‘What workflows are available’

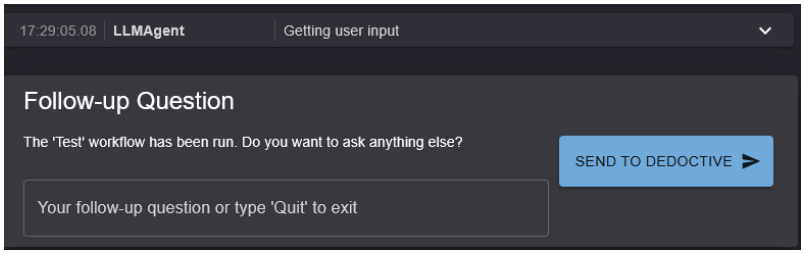
iii. The workflow then progresses until the ‘Continue’ userForm as shown below:



iv. We don't want to ask another question, so accept the default.
    1. Note that this form is defined in the ‘Test’ workflow

v. The workflow then completes, and returns its results to the LLMAgent

vi. The LLMAgent’s ad-hoc workflow then takes over the workflow, requesting if there is another follow-on task or should it quit.



vii. You can ask another question, or choose to type ‘Quit’ to exit
    1. The LLMAgent’s behaviour is controlled by the Instructions. It may be necessary to fine-tune the wording to get the behaviour desired. This behaviour will differ as well according to the chosen LLM model selected.

viii. All of this has been created with a single BPMN task: chat-in-a-box.

7.3. Takeaways

1. The deterministic nature of a BPMN workflow is both a strength and weakness.

2. BPMN's determinism is perfect when there is a well defined process that a business needs to follow to complete a process.
3. When a process is less well defined, or when the user might want to follow different paths through the process, then BPMN's determinism is a weakness.
4. The ability for a BPMN workflow to include an LLMAgent task has already been demonstrated. This tutorial demonstrates how even the interactions with a user can become a tool that the LLMAgent can invoke when it is necessary to get user input or answers that enable the process flow to continue. It is a chat-in-a-box.
5. The real world will never be at the extreme of BPMN's determinism, nor the LLMs ad-hoc process flows. So Dedoctive Deductions creates the perfect blend in which the perfect balance can be struck to automate complex workflows.
6. This is demonstrated by the ability for the LLMAgent to invoke other deterministic (aka BPMN) workflows, which might contain other LLMAgents with ad-hoc workflows or workflow tasks containing deterministic BPMN workflow.

8. Tutorial-8

8.1. Scope: Add dynamic elements to userForms

Having plunged into the depths of Dedoctive-Deductions and allowed an LLM to create a form, we might have concluded that a more deterministic approach would often be better, especially when creating a form.

The advantage of the LLM-generated UserForms is that they will adapt to prior circumstances within the session.Therefore this tutorial describes how UserForms can be dynamically adjusted to reflect the state of the workflow session.

The BPMN for this tutorial can be found in Tutorials/Tutorial-8

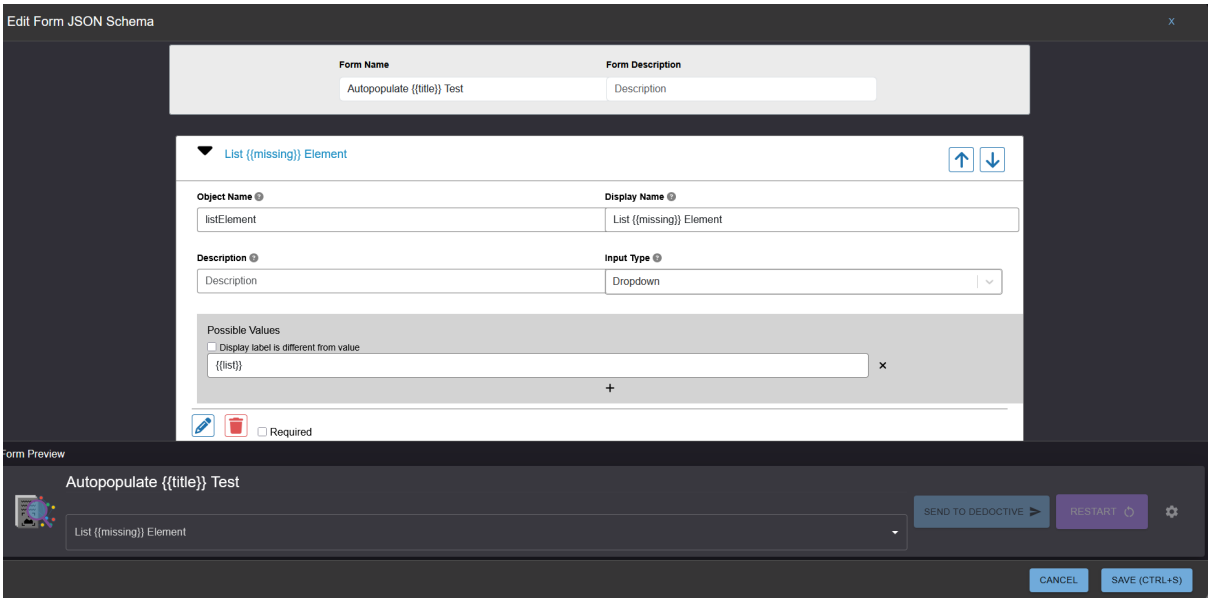
8.2. Steps

Open up Designer and create a new workflow, ***Tutorial-8*** with a single User task.

1. Each User Task has Pre/Post Scripts that allows workflow data to be created, updated, and deleted.
    a. The Pre script runs before the task runs and the Post script after
    b. Expand the Pre/Post scripts section in the property panel of the UserTask
    c. Enter the following as a Pre-Script:

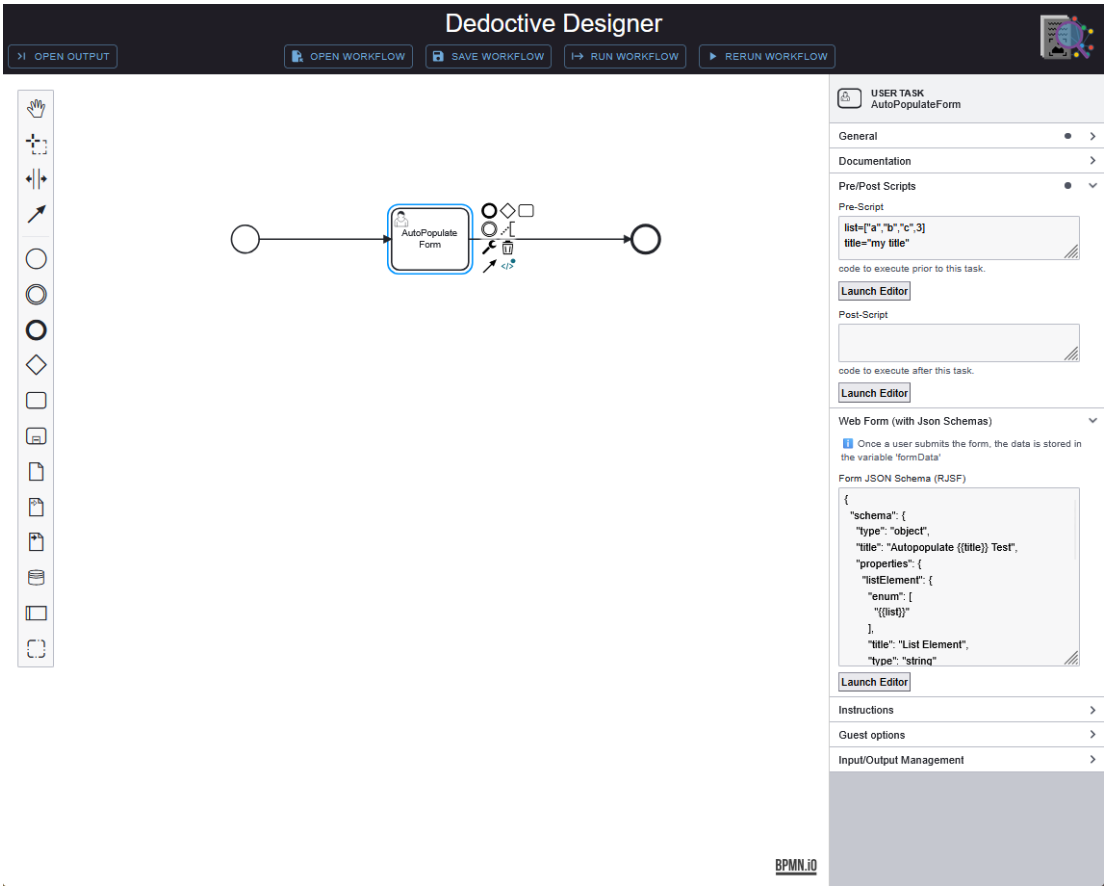
```
list=["a","b","c",3]
title="my title"
```

2. Expand the Web Form section and launch the Web Form editor, then enter the following form details:

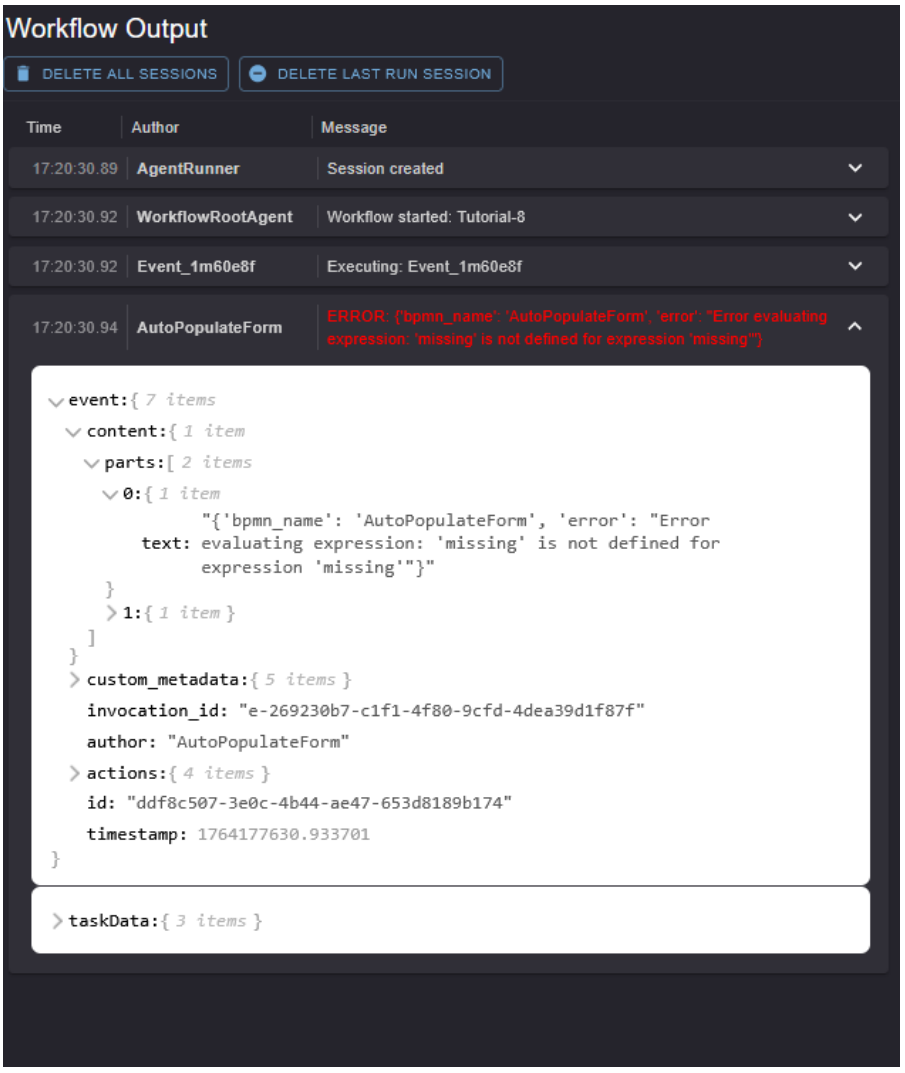


a. Note that the fields contain variables/expressions contained within {{...}}. These will be evaluated prior to the form being submitted for rendering.
    i. These expressions have access to any of the variables defined in the taskData.
b. Normally the results of the evaluation of the expression will be inserted within the text.
c. In the case of a 'list' expression, the values will be converted to a comma-separated list of values WITHOUT the enclosing '[...]'.
    i. This is so an expression can be used for a 'Dropdown' which expects a comma-separated list of strings

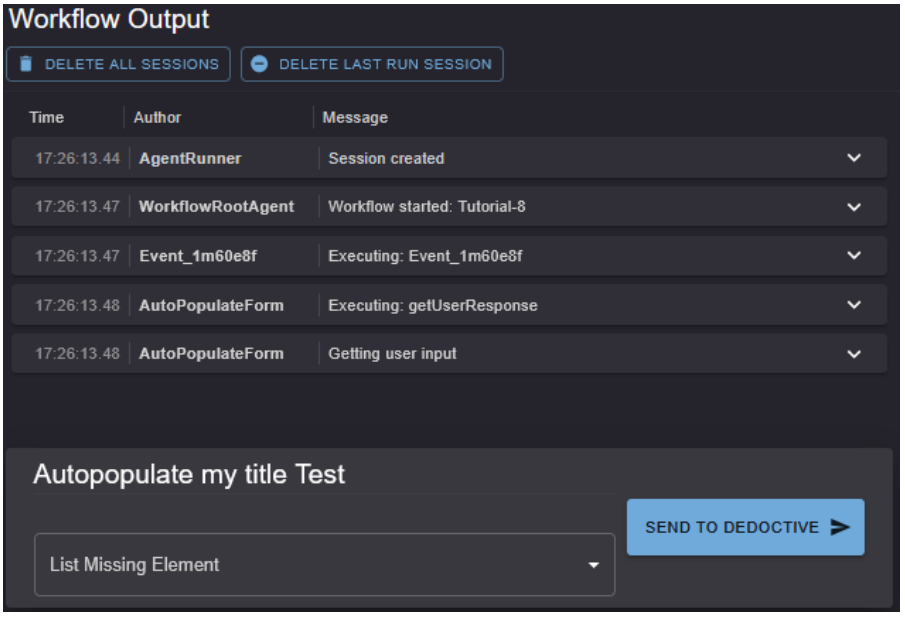
3. The resulting Designer should be as follows:



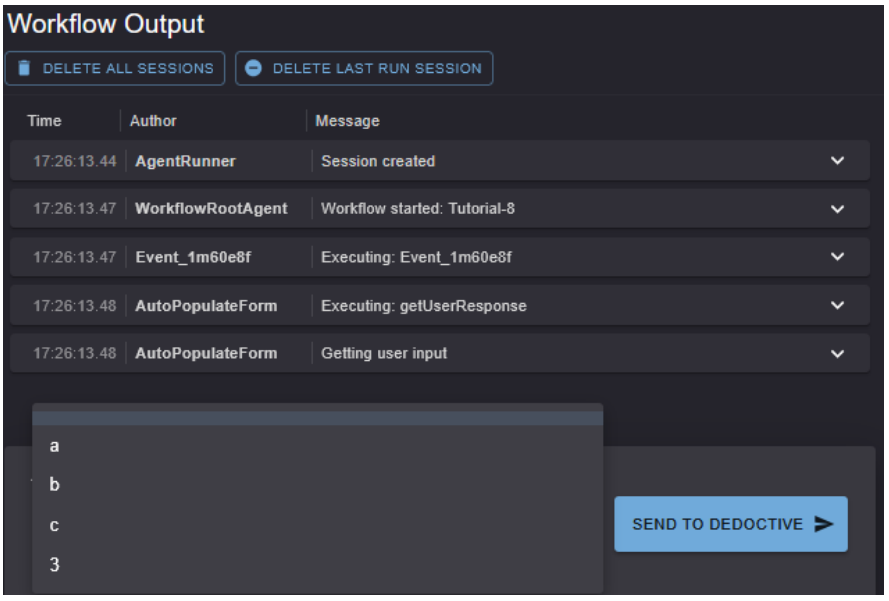
4. Run the workflow.
    a. However it will fail because the expression {{missing}} cannot be evaluated.
    b. As well as highlighting the error in the message, the details will always be found in event.content.parts



5. We can then correct the error by assigning a value to missing in the Pre-Script of the User task.
    a. Add this line to the end of the Pre-Script: ***missing = "Missing"***
    b. The user Form will appear with the {{...}} expressions substituted as shown below:



c. The dropdown contains a list constructed from the Python list as shown below:



8.3. Takeaways

1. Deterministic tasks can dynamically populate labels, lists etc within the user forms to reflect the progress of the workflow session up to that point.
2. This acts as a half-way between the ad-hoc forms that an LLMAgent can generate if it has access to the userTool, and the fixed forms typical of a BPMN workflow

9. Tutorial-9

9.1. Scope: Dynamic Debugging of workflows

Workflows are a very high-level programming language, even though they are more descriptive than procedural like regular languages. When developing a new workflow mistakes will be made, yes really!

Dedoctive-Designer does make it very easy to develop the workflows interactively, and quickly see the results in the Workflow output. Any errors in the workflow will appear within the Workflow Output panel in the Message associated with the task that caused the error. Additionally, the workflow diagram will highlight the task at which the execution of the workflow was halted.

Workflows can become much more intricate with subWorkflows, iterations, conditional paths and so on. So the visual debugging that Dedoctive-Designer offers becomes very important.

However the ability for Dedoctive-Deduction workflows to include agentic tasks, introduces another problem that many uses of agentic processes will have encountered: speed or lack of it. Response times from an LLM will take many seconds but they can even take many minutes when submitting a large context window or expecting a lengthy response. Thus, when a Deduction workflow includes such tasks, it makes debugging time consuming when one has to wait for a task, that's early in the workflow, to complete, before progressing (and debugging) subsequent tasks.

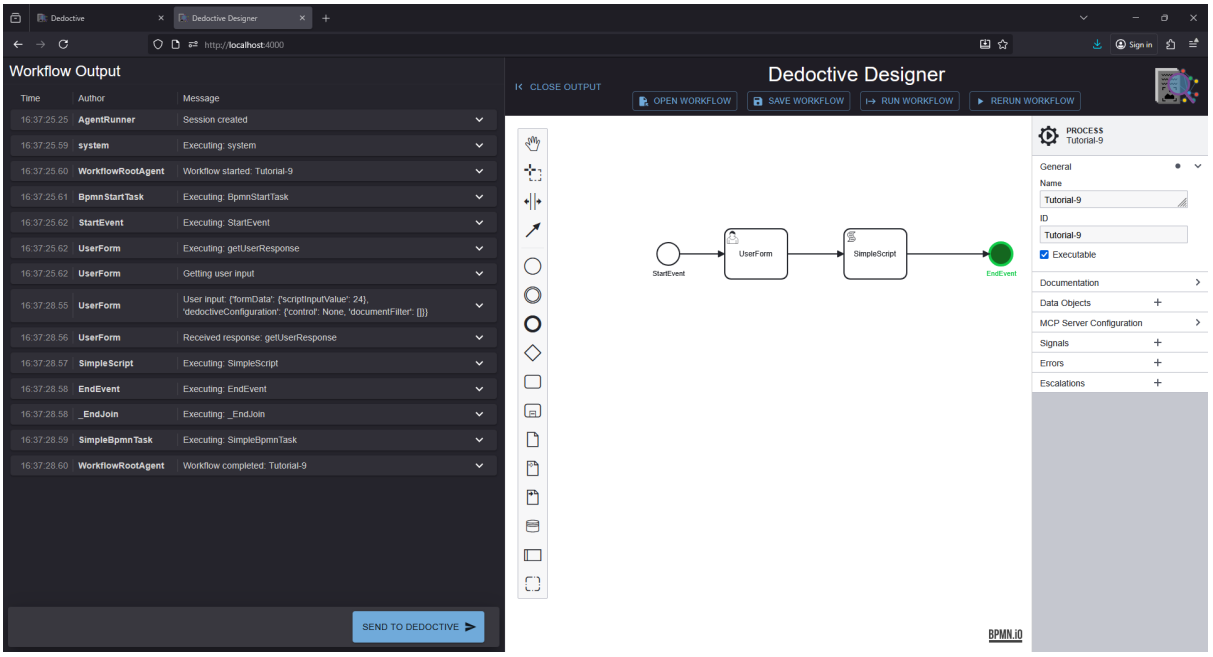
Dedoctive-Deduction and Dedoctive-Designer supports a unique concept: **DynamicDebugging**. With Dynamic-Debugging, a workflow can be 'rerun'. When a workflow is 'rerun' it will use the prior run's results for any task that is unchanged. It will only execute the tasks that have changed since the last run and insert the new results. It will also execute any task subsequent to a changed task.

Dynamic-Debugging therefore avoids the need to rerun time-consuming (and also expensive) tasks, in particular agentic tasks.

The BPMN for this tutorial can be found in Tutorials/Tutorial-9

9.2. Steps

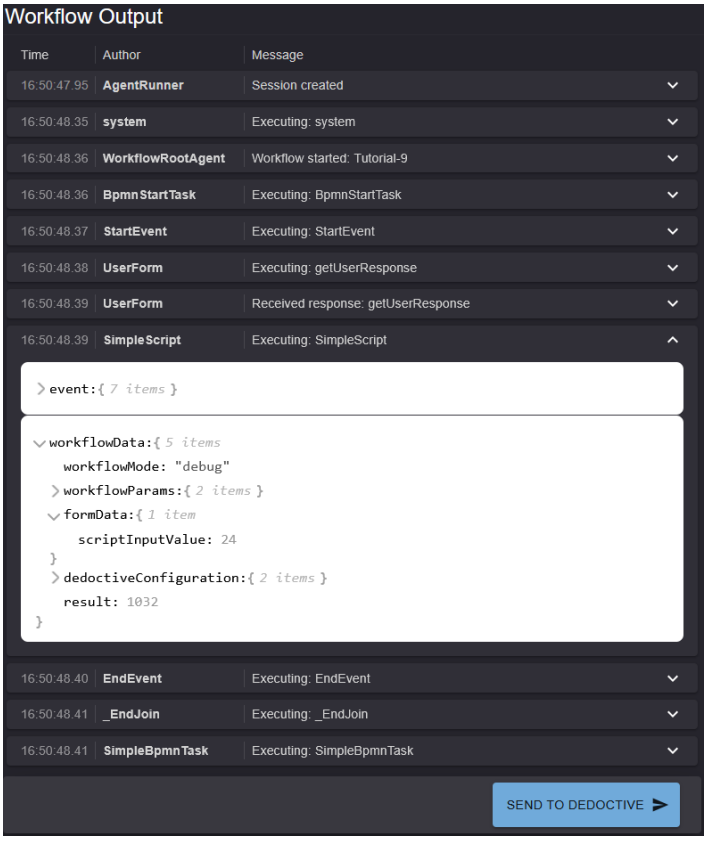
1. Create a new workflow.
    a. To make life easy start with Tutorial-2, a userForm followed by a script. This is sufficient to demonstrate Dynamic-Debugging
    b. For good housekeeping purpose, rename the overall workflows process name and id '***Tutorial-9***'
2. Run the workflow to completion
    a. Use the 'Run' button to run the workflow to completion
    b. This then means there is a 'last session' which we will be able to rerun



3. Edit the SimpleScript task to simulate a dynamic change to the workflow.
    a. Make sure you keep the UserForm unchanged
    b. Edit the SimpleScript's script in any way, for example change 42 to 43:

```
result = 43 * formData.get('scriptInputValue', 0)
```

4. Click 'Rerun' to run the workflow with dynamic debugging
    a. The workflow will run to completion *WITHOUT* requesting user input.
    b. This is because the userForm is unchanged, so Deduction has no need to rerun the userForm task
    c. It will use the same value as the user entered on the prior 'run'
    d. Examine the SimpleScript taskData:
        i. The results now shows 1032 (= 24 *43)



5. Experiment with other edits to the BPMN:
    a. Even changing the name of a task will cause it to be re-executed.
    b. Adding a new task after an existing task that previously ran successfully, will cause the prior task to be re-executed on a rerun as its definition within the overall workflow has changed.
    c. There is no need to ‘Run’ this workflow again. One can now keep on rerunning as long as you are content in using the prior results.

9.3. Takeaways

1. Dynamic-debugging is a unique feature of Dedoctive that greatly eases the rapid development of workflows even when the tasks are long running.
2. A rerun of a workflow will only re-execute a task that has been changed, instead using the saved results for prior steps in the workflow.

10. Tutorial-10

10.1. Scope: Globally scope variables and function definitions

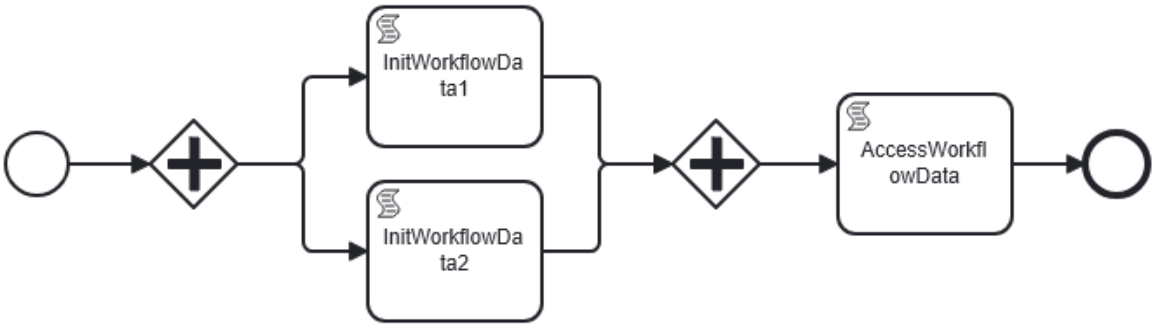
Within these tutorials the scripts to manipulate the task data have been kept deliberately simple so that the structure and organization of the workflow can be revealed. However, the strength of Dedoctive’s hybrid-agentic-BPMN workflows is that they can tackle very challenging data processing tasks. Consequently some scripts might become both complex and repeated throughout the workflow in different tasks.
Most data is stored within the ‘taskData’ dictionary. ‘taskData’ is passed from task-to-task, each task getting its own copy of ‘taskData’.
However, Dedoctive also supports a workflow-global dictionary named ‘workflowData’ for storing constants. ‘workflowData’ is not copied from task-to-task. Instead it is globally

available throughout the workflow. Thus, within this dictionary, variables that are required throughout the workflow, such as a JSONSchema specification, and function definitions, such as a JSONSchema preprocessing function can be stored.
The 'workflowData' dictionary needs to be initialised so that its variables and function definitions are available to subsequent workflow tasks. Note that 'workflowData should be **treated as immutable**: once a value has been set within the dictionary it should not be altered within a task later in the workflow. Note also that when a workflow is 'rerun' using the dynamic debugging feature of Deductive-Deduction-Designer, the workflowData is initialised with the **last** state of the workflowData.

This tutorial, found in Tutorial-10, demonstrates how workflowData can be initialized with variables and function definitions, which can be accessed later in the workflow.

10.2. Steps

1. This example workflow consists of 3 script tasks and 2 parallel gateways as shown below:



2. To place a parallel gateway, first add a gateway element from the palette on the left, select the placed gateway, click the spanner and choose "Parallel Gateway"
    a. Parallel gateways allow multiple tasks to be completed simultaneously.
    b. A parallel gateway must also be placed at the end of the split paths to merge them back together.
    c. This merging parallel gateway waits for all parallel tasks to complete before continuing on.
3. InitWorkflowData1 and InitWorkflowData2 assign values to the global workflowData
    a. The script for InitWorkflowData1 is:

```
workflowData['global1'] = 'global value 1'
```

    b. The script for InitWorkflowData2 is:

```
workflowData['global2'] = 'global value 2'
```

    c. AccessWorkflowData fetches those values and assigns global1 and global2 to task data:

```
global1 = workflowData['global1']
global2 = workflowData['global2']

workflow = workflowData
```

4. The last statement which assigns workflowData to a taskData variable called workflow is simply so the workflowData can be seen in the Workflow Output window.
5. If we run this workflow to completion, the taskData will be as shown below:

10.3. Takeaways

1. The global dictionary ‘workflowData’ is useful to hold global data, without the overhead of being copied to every taskData
2. The workflowData is not normally visible within the Workflow Output as this only shows event data associated with each task. The workflowData can be seen if it is assigned to a taskData variable
3. A use-case of workflowData is when we wish to hold a very large JSONSchema but only want to use fragments of that universal schema in tasks within the workflow.