Handoffs

Handoffs allow an agent to delegate tasks to another agent. This is particularly useful in scenarios where different agents specialize in distinct areas. For example, a customer support app might have agents that each specifically handle tasks like order status, refunds, FAQs, etc.

Handoffs are represented as tools to the LLM. So if there's a handoff to an agent named `Refund Agent`, the tool would be called `transfer_to_refund_agent`.

Creating a handoff

All agents have a `handoffs` param, which can either take an `Agent` directly, or a `Handoff` object that customizes the Handoff.

You can create a handoff using the `handoff()` function provided by the Agents SDK. This function allows you to specify the agent to hand off to, along with optional overrides and input filters.

Basic Usage

Here's how you can create a simple handoff:

```
from agents import Agent, handoff

billing_agent = Agent(name="Billing agent")
refund_agent = Agent(name="Refund agent")

 1
triage_agent = Agent(name="Triage agent", handoffs=[billing_agent,
handoff(refund_agent)])
```

1 You can use the agent directly (as in `billing_agent`), or you can use the `handoff()` function.

Customizing handoffs via the `handoff()` function

The `handoff()` function lets you customize things.

- `agent`: This is the agent to which things will be handed off.
- `tool_name_override`: By default, the `Handoff.default_tool_name()` function is used, which resolves to `transfer_to_<agent_name>`. You can override this.
- `tool_description_override`: Override the default tool description from `Handoff.default_tool_description()`

- `on_handoff`: A callback function executed when the handoff is invoked. This is useful for things like kicking off some data fetching as soon as you know a handoff is being invoked. This function receives the agent context, and can optionally also receive LLM generated input. The input data is controlled by the `input_type` param.
- `input_type`: The type of input expected by the handoff (optional).
- `input_filter`: This lets you filter the input received by the next agent. See below for more.

```
from agents import Agent, handoff, RunContextWrapper

def on_handoff(ctx: RunContextWrapper[None]):
    print("Handoff called")

agent = Agent(name="My agent")

handoff_obj = handoff(
    agent=agent,
    on_handoff=on_handoff,
    tool_name_override="custom_handoff_tool",
    tool_description_override="Custom description",
)
```

Handoff inputs

In certain situations, you want the LLM to provide some data when it calls a handoff. For example, imagine a handoff to an "Escalation agent". You might want a reason to be provided, so you can log it.

```
from pydantic import BaseModel

from agents import Agent, handoff, RunContextWrapper

class EscalationData(BaseModel):
    reason: str

async def on_handoff(ctx: RunContextWrapper[None], input_data: EscalationData):
    print(f"Escalation agent called with reason: {input_data.reason}")

agent = Agent(name="Escalation agent")

handoff_obj = handoff(
    agent=agent,
    on_handoff=on_handoff,
    input_type=EscalationData,
)
```

Input filters

When a handoff occurs, it's as though the new agent takes over the conversation, and gets to see the entire previous conversation history. If you want to change this, you can set an `input_filter`. An input filter is a function that receives the existing input via a `HandoffInputData`, and must return a new `HandoffInputData`.

There are some common patterns (for example removing all tool calls from the history), which are implemented for you in `agents.extensions.handoff_filters`

```
from agents import Agent, handoff
from agents.extensions import handoff_filters

agent = Agent(name="FAQ agent")

handoff_obj = handoff(
    agent=agent,
    input_filter=handoff_filters.remove_all_tools, (1)
)
```

(1) This will automatically remove all tools from the history when `FAQ agent` is called.

Recommended prompts

To make sure that LLMs understand handoffs properly, we recommend including information about handoffs in your agents. We have a suggested prefix in `agents.extensions.handoff_prompt.RECOMMENDED_PROMPT_PREFIX`, or you can call `agents.extensions.handoff_prompt.prompt_with_handoff_instructions` to automatically add recommended data to your prompts.

```
from agents import Agent
from agents.extensions.handoff_prompt import RECOMMENDED_PROMPT_PREFIX

billing_agent = Agent(
    name="Billing agent",
    instructions=f"""{RECOMMENDED_PROMPT_PREFIX}
    <Fill in the rest of your prompt here>.""",
)
```