

[Modules](#)[More](#)[Callbacks](#)[Multiple callback handlers](#)

# Multiple callback handlers

In the previous examples, we passed in callback handlers upon creation of an object by using `callbacks=`. In this case, the callbacks will be scoped to that particular object.

However, in many cases, it is advantageous to pass in handlers instead when running the object. When we pass through `CallbackHandlers` using the `callbacks` keyword arg when executing an run, those callbacks will be issued by all nested objects involved in the execution. For example, when a handler is passed through to an `Agent`, it will be used for all callbacks related to the agent and all the objects involved in the agent's execution, in this case, the `Tools`, `LLMChain`, and `LLM`.

This prevents us from having to manually attach the handlers to each individual nested object.

```
from typing import Any, Dict, List, Union

from langchain.agents import AgentType,
initialize_agent, load_tools
from langchain.callbacks.base import
```

```
BaseCallbackHandler
```

```
from langchain.schema import AgentAction
from langchain_openai import OpenAI
```

```
# First, define custom callback handler
implementations
```

```
class
```

```
MyCustomHandlerOne(BaseCallbackHandler):
```

```
    def on_llm_start(
        self, serialized: Dict[str, Any],
        prompts: List[str], **kwargs: Any
    ) -> Any:
        print(f"on_llm_start
{serialized['name']}")
```

```
    def on_llm_new_token(self, token: str,
        **kwargs: Any) -> Any:
        print(f"on_new_token {token}")
```

```
    def on_llm_error(
        self, error: Union[Exception,
        KeyboardInterrupt], **kwargs: Any
    ) -> Any:
        """Run when LLM errors."""
```

```
    def on_chain_start(
        self, serialized: Dict[str, Any],
        inputs: Dict[str, Any], **kwargs: Any
    ) -> Any:
        print(f"on_chain_start
```

```
{serialized['name']}]}")

    def on_tool_start(
        self, serialized: Dict[str, Any],
input_str: str, **kwargs: Any
    ) -> Any:
        print(f"on_tool_start
{serialized['name']}]")

    def on_agent_action(self, action:
AgentAction, **kwargs: Any) -> Any:
        print(f"on_agent_action {action}")

class
MyCustomHandlerTwo(BaseCallbackHandler):
    def on_llm_start(
        self, serialized: Dict[str, Any],
prompts: List[str], **kwargs: Any
    ) -> Any:
        print(f"on_llm_start (I'm the second
handler!!) {serialized['name']}]")

# Instantiate the handlers
handler1 = MyCustomHandlerOne()
handler2 = MyCustomHandlerTwo()

# Setup the agent. Only the `llm` will issue
callbacks for handler2
llm = OpenAI(temperature=0, streaming=True,
```

```

callbacks=[handler2])
tools = load_tools(["llm-math"], llm=llm)
agent = initialize_agent(tools, llm,
agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION)

# Callbacks for handler1 will be issued by
every object involved in the
# Agent execution (llm, llmchain, tool, agent
executor)
agent.run("What is 2 raised to the 0.235
power?", callbacks=[handler1])

```

```

on_chain_start AgentExecutor
on_chain_start LLMChain
on_llm_start OpenAI
on_llm_start (I'm the second handler!!)
OpenAI
on_new_token I
on_new_token need
on_new_token to
on_new_token use
on_new_token a
on_new_token calculator
on_new_token to
on_new_token solve
on_new_token this
on_new_token .
on_new_token
Action
on_new_token :

```

```
on_new_token    Calculator
on_new_token
Action
on_new_token    Input
on_new_token    :
on_new_token    2
on_new_token    ^
on_new_token    0
on_new_token    .
on_new_token    235
on_new_token
on_agent_action
AgentAction(tool='Calculator',
tool_input='2^0.235', log=' I need to use a
calculator to solve this.\nAction:
Calculator\nAction Input: 2^0.235')
on_tool_start Calculator
on_chain_start LLMMathChain
on_chain_start LLMChain
on_llm_start OpenAI
on_llm_start (I'm the second handler!!)
OpenAI
on_new_token
on_new_token    ``text
on_new_token

on_new_token    2
on_new_token    **
on_new_token    0
on_new_token    .
on_new_token    235
```

```
on_new_token
```

```
on_new_token ```
```

```
on_new_token ...
```

```
on_new_token num
```

```
on_new_token expr
```

```
on_new_token .
```

```
on_new_token evaluate
```

```
on_new_token ("
```

```
on_new_token 2
```

```
on_new_token **
```

```
on_new_token 0
```

```
on_new_token .
```

```
on_new_token 235
```

```
on_new_token ")
```

```
on_new_token ...
```

```
on_new_token
```

```
on_new_token
```

```
on_chain_start LLMChain
```

```
on_llm_start OpenAI
```

```
on_llm_start (I'm the second handler!!)
```

```
OpenAI
```

```
on_new_token I
```

```
on_new_token now
```

```
on_new_token know
```

```
on_new_token the
```

```
on_new_token final
```

```
on_new_token answer
```

```
on_new_token .
```

```
on_new_token
```

```
Final
```

```
on_new_token  Answer
```

```
on_new_token  :
```

```
on_new_token  1
```

```
on_new_token  .
```

```
on_new_token  17
```

```
on_new_token  690
```

```
on_new_token  67
```

```
on_new_token  372
```

```
on_new_token  187
```

```
on_new_token  674
```

```
on_new_token
```

```
'1.1769067372187674'
```