Messages and chat history

PydanticAI provides access to messages exchanged during an agent run. These messages can be used both to continue a coherent conversation, and to understand how an agent performed.

Accessing Messages from Results

After running an agent, you can access the messages exchanged during that run from the `result` object.

Both `RunResult` (returned by `Agent.run`, `Agent.run_sync`) and `StreamedRunResult` (returned by `Agent.run_stream`) have the following methods:

- `all_messages()`: returns all messages, including messages from prior runs and system prompts. There's also a variant that returns JSON bytes, `all_messages_json()`.
- `new_messages()`: returns only the messages from the current run, excluding system prompts, this is generally the data you want when you want to use the messages in further runs to continue the conversation. There's also a variant that returns JSON bytes, `new_messages_json()`.

StreamedRunResult and complete messages

On `StreamedRunResult`, the messages returned from these methods will only include the final result message once the stream has finished.

E.g. you've awaited one of the following coroutines:

- `StreamedRunResult.stream()`
- `StreamedRunResult.stream_text()`
- `StreamedRunResult.stream_structured()`
- `StreamedRunResult.get_data()`

Note: The final result message will NOT be added to result messages if you use `.stream_text(delta=True)` since in this case the result content is never built as one string.

Example of accessing methods on a `RunResult`:

run_result_messages.py

```python
from pydantic_ai import Agent

agent = Agent('openai:gpt-4o', system_prompt='Be a helpful assistant.')

result = agent.run_sync('Tell me a joke.')
print(result.data)
#> Did you hear about the toothpaste scandal? They called it Colgate.

# all messages from the run
print(result.all_messages())
"""
[
    SystemPrompt(content='Be a helpful assistant.', role='system'),
    UserPrompt(
        content='Tell me a joke.',
        timestamp=datetime.datetime(...),
        role='user',
    ),
    ModelTextResponse(
        content='Did you hear about the toothpaste scandal? They called it Colgate.',
        timestamp=datetime.datetime(...),
        role='model-text-response',
    ),
]
"""

# messages excluding system prompts
print(result.new_messages())
"""
[
    UserPrompt(
        content='Tell me a joke.',
        timestamp=datetime.datetime(...),
        role='user',
    ),
    ModelTextResponse(
        content='Did you hear about the toothpaste scandal? They called it Colgate.',
        timestamp=datetime.datetime(...),
        role='model-text-response',
    ),
]
"""
```

(This example is complete, it can be run "as is")

Example of accessing methods on a `StreamedRunResult`:

streamed_run_result_messages.py

```python
from pydantic_ai import Agent

agent = Agent('openai:gpt-4o', system_prompt='Be a helpful assistant.')


async def main():
    async with agent.run_stream('Tell me a joke.') as result:
        # incomplete messages before the stream finishes
        print(result.all_messages())
        """
        [
            SystemPrompt(content='Be a helpful assistant.', role='system'),
            UserPrompt(
                content='Tell me a joke.',
                timestamp=datetime.datetime(...),
                role='user',
            ),
        ]
        """

        async for text in result.stream():
            print(text)
            #> Did you hear
            #> Did you hear about the toothpaste
```

```
            #> Did you hear about the toothpaste scandal? They called
            #> Did you hear about the toothpaste scandal? They called it Colgate.

        # complete messages once the stream finishes
        print(result.all_messages())
        """
        [
            SystemPrompt(content='Be a helpful assistant.', role='system'),
            UserPrompt(
                content='Tell me a joke.',
                timestamp=datetime.datetime(...),
                role='user',
            ),
            ModelTextResponse(
                content='Did you hear about the toothpaste scandal? They called it Colgate.',
                timestamp=datetime.datetime(...),
                role='model-text-response',
            ),
        ]
        """
```

(This example is complete, it can be run "as is")

Using Messages as Input for Further Agent Runs

The primary use of message histories in PydanticAI is to maintain context across multiple agent runs.

To use existing messages in a run, pass them to the `message_history` parameter of `Agent.run`, `Agent.run_sync` or `Agent.run_stream`.

`all_messages()` vs. `new_messages()`

PydanticAI will inspect any messages it receives for system prompts.

If any system prompts are found in `message_history`, new system prompts are not generated, otherwise new system prompts are generated and inserted before `message_history` in the list of messages used in the run.

Thus you can decide whether you want to use system prompts from a previous run or generate them again by using `all_messages()` or `new_messages()`.

Reusing messages in a conversation

```
from pydantic_ai import Agent

agent = Agent('openai:gpt-4o', system_prompt='Be a helpful assistant.')

result1 = agent.run_sync('Tell me a joke.')
print(result1.data)
#> Did you hear about the toothpaste scandal? They called it Colgate.

result2 = agent.run_sync('Explain?', message_history=result1.new_messages())
print(result2.data)
#> This is an excellent joke invent by Samuel Colvin, it needs no explanation.

print(result2.all_messages())
"""
[
    SystemPrompt(content='Be a helpful assistant.', role='system'),
    UserPrompt(
        content='Tell me a joke.',
        timestamp=datetime.datetime(...),
        role='user',
    ),
    ModelTextResponse(
        content='Did you hear about the toothpaste scandal? They called it Colgate.',
        timestamp=datetime.datetime(...),
        role='model-text-response',
    ),
    UserPrompt(
        content='Explain?',
        timestamp=datetime.datetime(...),
        role='user',
    ),
    ModelTextResponse(
        content='This is an excellent joke invent by Samuel Colvin, it needs no explanation.',
        timestamp=datetime.datetime(...),
        role='model-text-response',
    ),
]
"""
```

(This example is complete, it can be run "as is")

Other ways of using messages

Since messages are defined by simple dataclasses, you can manually create and manipulate, e.g. for testing.

The message format is independent of the model used, so you can use messages in different agents, or the same agent with different models.

```
from pydantic_ai import Agent

agent = Agent('openai:gpt-4o', system_prompt='Be a helpful assistant.')

result1 = agent.run_sync('Tell me a joke.')
print(result1.data)
#> Did you hear about the toothpaste scandal? They called it Colgate.

result2 = agent.run_sync(
    'Explain?', model='gemini-1.5-pro', message_history=result1.new_messages()
)
print(result2.data)
#> This is an excellent joke invent by Samuel Colvin, it needs no explanation.

print(result2.all_messages())
"""
[
    SystemPrompt(content='Be a helpful assistant.', role='system'),
    UserPrompt(
        content='Tell me a joke.',
        timestamp=datetime.datetime(...),
        role='user',
    ),
    ModelTextResponse(
        content='Did you hear about the toothpaste scandal? They called it Colgate.',
        timestamp=datetime.datetime(...),
        role='model-text-response',
```

```
    ),
    UserPrompt(
        content='Explain?',
        timestamp=datetime.datetime(...),
        role='user',
    ),
    ModelTextResponse(
        content='This is an excellent joke invent by Samuel Colvin, it needs no explanation.',
        timestamp=datetime.datetime(...),
        role='model-text-response',
    ),
]
"""
```

Examples

For a more complete example of using messages in conversations, see the chat app example.