

## pydantic\_ai.messages

### Message module-attribute

```

Message = Union[
    SystemPrompt,
    UserPrompt,
    ToolReturn,
    RetryPrompt,
    ModelTextResponse,
    ModelStructuredResponse,
]

```

Any message send to or returned by a model.

### SystemPrompt dataclass

A system prompt, generally written by the application developer.

This gives the model context and guidance on how to respond.

```

99 Source code in pydantic_ai_slim/pydantic_ai/messages.py
15 @dataclass
16 class SystemPrompt:
17     """A system prompt, generally written by the application developer.
18
19     This gives the model context and guidance on how to respond.
20     """
21
22     content: str
23     """The content of the prompt."""
24     role: Literal['system'] = 'system'
25     """Message type identifier, this type is available on all message as a discriminator."""

```

#### content instance-attribute

```
content: str
```

The content of the prompt.

#### role class-attribute instance-attribute

```
role: Literal['system'] = 'system'
```

Message type identifier, this type is available on all message as a discriminator.

### UserPrompt dataclass

A user prompt, generally written by the end user.

Content comes from the `user_prompt` parameter of `Agent.run`, `Agent.run_sync`, and `Agent.run_stream`.

```

99 Source code in pydantic_ai_slim/pydantic_ai/messages.py
28 @dataclass
29 class UserPrompt:
30     """A user prompt, generally written by the end user.
31
32     Content comes from the 'user_prompt' parameter of ['Agent.run'] [pydantic_ai.Agent.run],
33     ['Agent.run_sync'] [pydantic_ai.Agent.run_sync], and ['Agent.run_stream'] [pydantic_ai.Agent.run_stream].
34     """
35
36     content: str
37     """The content of the prompt."""
38     timestamp: datetime = field(default_factory=now_utc)
39     """The timestamp of the prompt."""
40     role: Literal['user'] = 'user'
41     """Message type identifier, this type is available on all message as a discriminator."""

```

#### content instance-attribute

```
content: str
```

The content of the prompt.

#### timestamp class-attribute instance-attribute

```
timestamp: datetime = field(default_factory=now_utc)
```

The timestamp of the prompt.

#### role class-attribute instance-attribute

```
role: Literal['user'] = 'user'
```

Message type identifier, this type is available on all message as a discriminator.

### ToolReturn dataclass

A tool return message, this encodes the result of running a tool.

```

47 @dataclass
48 class ToolReturn:
49     """A tool return message, this encodes the result of running a tool."""
50
51     tool_name: str
52     """The name of the "tool" was called."""
53     content: Any
54     """The return value."""
55     tool_call_id: str | None = None
56     """Optional tool call identifier, this is used by some models including OpenAI."""
57     timestamp: datetime = field(default_factory=now_utc)
58     """The timestamp, when the tool returned."""
59     role: Literal['tool-return'] = 'tool-return'
60     """Message type identifier, this type is available on all message as a discriminator."""
61
62     def model_response_str(self) -> str:
63         if isinstance(self.content, str):
64             return self.content
65         else:
66             return tool_return_ta.dump_json(self.content).decode()
67
68     def model_response_object(self) -> dict[str, Any]:
69         # gemini supports JSON dict return values, but no other JSON types, hence we wrap anything else in a dict
70         if isinstance(self.content, dict):
71             return tool_return_ta.dump_python(self.content, mode='json') # pyright: ignore[reportUnknownMemberType]
72         else:
73             return {'return_value': tool_return_ta.dump_python(self.content, mode='json')}

```

**tool\_name** instance-attribute

```
tool_name: str
```

The name of the "tool" was called.

**content** instance-attribute

```
content: Any
```

The return value.

**tool\_call\_id** class-attribute instance-attribute

```
tool_call_id: str | None = None
```

Optional tool call identifier, this is used by some models including OpenAI.

**timestamp** class-attribute instance-attribute

```
timestamp: datetime = field(default_factory=now_utc)
```

The timestamp, when the tool returned.

**role** class-attribute instance-attribute

```
role: Literal['tool-return'] = 'tool-return'
```

Message type identifier, this type is available on all message as a discriminator.

**RetryPrompt** dataclass

A message back to a model asking it to try again.

This can be sent for a number of reasons:

- Pydantic validation of tool arguments failed, here content is derived from a Pydantic `ValidationError`
- a tool raised a `ModelRetry` exception
- no tool was found for the tool name
- the model returned plain text when a structured response was expected
- Pydantic validation of a structured response failed, here content is derived from a Pydantic `ValidationError`
- a result validator raised a `ModelRetry` exception

```

79 @dataclass
80 class RetryPrompt:
81     """A message back to a model asking it to try again.
82
83     This can be sent for a number of reasons:
84
85     * Pydantic validation of tool arguments failed, here content is derived from a Pydantic
86       ['ValidationError'][pydantic_core.ValidationError]
87     * a tool raised a ['ModelRetry'][pydantic_ai.exceptions.ModelRetry] exception
88     * no tool was found for the tool name
89     * the model returned plain text when a structured response was expected
90     * Pydantic validation of a structured response failed, here content is derived from a Pydantic
91       ['ValidationError'][pydantic_core.ValidationError]
92     * a result validator raised a ['ModelRetry'][pydantic_ai.exceptions.ModelRetry] exception
93     """
94
95     content: list[pydantic_core.ErrorDetails] | str
96     """Details of why and how the model should retry.
97
98     If the retry was triggered by a ['ValidationError'][pydantic_core.ValidationError], this will be a list of
99     error details.
100     """
101     tool_name: str | None = None
102     """The name of the tool that was called, if any."""
103     tool_call_id: str | None = None
104     """Optional tool call identifier, this is used by some models including OpenAI."""
105     timestamp: datetime = field(default_factory=now_utc)
106     """The timestamp, when the retry was triggered."""
107     role: Literal['retry-prompt'] = 'retry-prompt'
108     """Message type identifier, this type is available on all message as a discriminator."""
109
110     def model_response(self) -> str:
111         if isinstance(self.content, str):
112             description = self.content
113         else:
114             json_errors = ErrorDetailsTa.dump_json(self.content, exclude={'__all__': {'ctx'}}), indent=2)
115             description = f'{len(self.content)} validation errors: {json_errors.decode()}'
116         return f'{description}\n\nFix the errors and try again.'

```

**content** instance-attribute

```
content: list[ErrorDetails] | str
```

Details of why and how the model should retry.

If the retry was triggered by a `ValidationError`, this will be a list of error details.

**tool\_name** class-attribute instance-attribute

```
tool_name: str | None = None
```

The name of the tool that was called, if any.

**tool\_call\_id** class-attribute instance-attribute

```
tool_call_id: str | None = None
```

Optional tool call identifier, this is used by some models including OpenAI.

**timestamp** class-attribute instance-attribute

```
timestamp: datetime = field(default_factory=now_utc)
```

The timestamp, when the retry was triggered.

**role** class-attribute instance-attribute

```
role: Literal['retry-prompt'] = 'retry-prompt'
```

Message type identifier, this type is available on all message as a discriminator.

**ModelAnyResponse** module-attribute

```

ModelAnyResponse = Union[
    ModelTextResponse, ModelStructuredResponse
]

```

Any response from a model.

**ModelTextResponse** dataclass

A plain text response from a model.

```

119 @dataclass
120 class ModelTextResponse:
121     """A plain text response from a model."""
122
123     content: str
124     """The text content of the response."""
125     timestamp: datetime = field(default_factory=now_utc)
126     """The timestamp of the response.
127
128     If the model provides a timestamp in the response (as OpenAI does) that will be used.
129     """
130     role: Literal['model-text-response'] = 'model-text-response'
131     """Message type identifier, this type is available on all message as a discriminator."""

```

**content** instance-attribute

```
content: str
```

The text content of the response.

**timestamp** class-attribute instance-attribute

```
timestamp: datetime = field(default_factory=now_utc)
```

The timestamp of the response.

If the model provides a timestamp in the response (as OpenAI does) that will be used.

**role** class-attribute instance-attribute

```
role: Literal["model-text-response"] = "model-text-response"
```

Message type identifier, this type is available on all message as a discriminator.

**ModelStructuredResponse** dataclass

A structured response from a model.

This is used either to call a tool or to return a structured response from an agent run.

Source code in `pydantic_ai_slim/pydantic_ai/messages.py`

```
179 @dataclass
180 class ModelStructuredResponse:
181     """A structured response from a model.
182
183     This is used either to call a tool or to return a structured response from an agent run.
184     """
185
186     calls: list[ToolCall]
187     """The tool calls being made."""
188     timestamp: datetime = field(default_factory=now_utc)
189     """The timestamp of the response.
190
191     If the model provides a timestamp in the response (as OpenAI does) that will be used.
192     """
193     role: Literal["model-structured-response"] = "model-structured-response"
194     """Message type identifier, this type is available on all message as a discriminator."""
```

**calls** instance-attribute

```
calls: list[ToolCall]
```

The tool calls being made.

**timestamp** class-attribute instance-attribute

```
timestamp: datetime = field(default_factory=now_utc)
```

The timestamp of the response.

If the model provides a timestamp in the response (as OpenAI does) that will be used.

**role** class-attribute instance-attribute

```
role: Literal["model-structured-response"] = (
    "model-structured-response"
)
```

Message type identifier, this type is available on all message as a discriminator.

**ToolCall** dataclass

Either a tool call from the agent.

Source code in `pydantic_ai_slim/pydantic_ai/messages.py`

```
150 @dataclass
151 class ToolCall:
152     """Either a tool call from the agent."""
153
154     tool_name: str
155     """The name of the tool to call."""
156     args: ArgsJson | ArgsDict
157     """The arguments to pass to the tool.
158
159     Either as JSON or a Python dictionary depending on how data was returned.
160     """
161     tool_call_id: str | None = None
162     """Optional tool call identifier, this is used by some models including OpenAI."""
163
164     @classmethod
165     def from_json(cls, tool_name: str, args_json: str, tool_call_id: str | None = None) -> ToolCall:
166         return cls(tool_name, ArgsJson(args_json), tool_call_id)
167
168     @classmethod
169     def from_dict(cls, tool_name: str, args_dict: dict[str, Any]) -> ToolCall:
170         return cls(tool_name, ArgsDict(args_dict))
171
172     def has_content(self) -> bool:
173         if isinstance(self.args, ArgsDict):
174             return any(self.args.args_dict.values())
175         else:
176             return bool(self.args.args_json)
```

**tool\_name** instance-attribute

```
tool_name: str
```

The name of the tool to call.

**args** instance-attribute

```
args: ArgsJson | ArgsDict
```

The arguments to pass to the tool.

Either as JSON or a Python dictionary depending on how data was returned.

**tool\_call\_id** class-attribute instance-attribute

```
tool_call_id: str | None = None
```

Optional tool call identifier, this is used by some models including OpenAI.

**ArgsJson** dataclass

Tool arguments as a JSON string.

Source code in `pydantic_ai_slim/pydantic_ai/messages.py`

```
134 @dataclass
135 class ArgsJson:
136     """Tool arguments as a JSON string."""
137
138     args_json: str
139     """A JSON string of arguments."""
```

**args\_json** instance-attribute

```
args_json: str
```

A JSON string of arguments.

**MessageTypeAdapter** module-attribute

```
MessageTypeAdapter = LazyTypeAdapter(
    list[Annotated[Message, Field(discriminator="role")]]
)
```

Pydantic **TypeAdapter** for (de)serializing messages.