On this page

agentchat.groupchat

GroupChat Objects

```
@dataclass
class GroupChat()
```

(In preview) A group chat class that contains the following data fields:

- agents: a list of participating agents.
- messages: a list of messages in the group chat.
- max_round: the maximum number of rounds.
- admin_name: the name of the admin agent if there is one. Default is "Admin". KeyBoardInterrupt will make the admin agent take over.
- func_call_filter: whether to enforce function call filter. Default is True. When set to True and when a message is a function call suggestion, the next speaker will be chosen from an agent which contains the corresponding function name in its `function_map`.
- speaker_selection_method: the method for selecting the next speaker. Default is "auto". Could be any of the following (case insensitive), will raise ValueError if not recognized:
  - "auto": the next speaker is selected automatically by LLM.
  - "manual": the next speaker is selected manually by user input.
  - "random": the next speaker is selected randomly.
  - "round_robin": the next speaker is selected in a round robin fashion, i.e., iterating in the same order as provided in `agents`.
- allow_repeat_speaker: whether to allow the same speaker to speak consecutively. Default is True, in which case all speakers are allowed to speak consecutively. If allow_repeat_speaker is a list of Agents, then only those listed agents are allowed to repeat. If set to False, then no speakers are allowed to repeat.

agent_names

```
@property
def agent_names() -> List[str]
```

Return the names of the agents in the group chat.

reset

```
def reset()
```

Reset the group chat.

append

```
def append(message: Dict, speaker: Agent)
```

Append a message to the group chat. We cast the content to str here so that it can be managed by text-based model.

agent_by_name

```
def agent_by_name(name: str) -> Agent
```

Returns the agent with a given name.

next_agent

```
def next_agent(agent: Agent, agents: Optional[List[Agent]] = None) -> Agent
```

Return the next agent in the list.

select_speaker_msg

```
def select_speaker_msg(agents: Optional[List[Agent]] = None) -> str
```

Return the system message for selecting the next speaker. This is always the *first* message in the context.

select_speaker_prompt

```
def select_speaker_prompt(agents: Optional[List[Agent]] = None) -> str
```

Return the floating system prompt selecting the next speaker. This is always the *last* message in the context.

manual_select_speaker

```
def manual_select_speaker(
        agents: Optional[List[Agent]] = None) -> Union[Agent, None]
```

Manually select the next speaker.

select_speaker

```
def select_speaker(last_speaker: Agent, selector: ConversableAgent)
```

Select the next speaker.

a_select_speaker

```
async def a_select_speaker(last_speaker: Agent, selector: ConversableAgent)
```

Select the next speaker.

GroupChatManager Objects

```
class GroupChatManager(ConversableAgent)
```

(In preview) A chat manager agent that can manage a group chat of multiple agents.

run_chat

```
def run_chat(messages: Optional[List[Dict]] = None,
             sender: Optional[Agent] = None,
             config: Optional[GroupChat] = None) -> Union[str, Dict, None]
```

Run a group chat.

a_run_chat

```
async def a_run_chat(messages: Optional[List[Dict]] = None,
                     sender: Optional[Agent] = None,
                     config: Optional[GroupChat] = None)
```

Run a group chat asynchronously.

Previous
« conversable_agent

Next
user_proxy_agent »