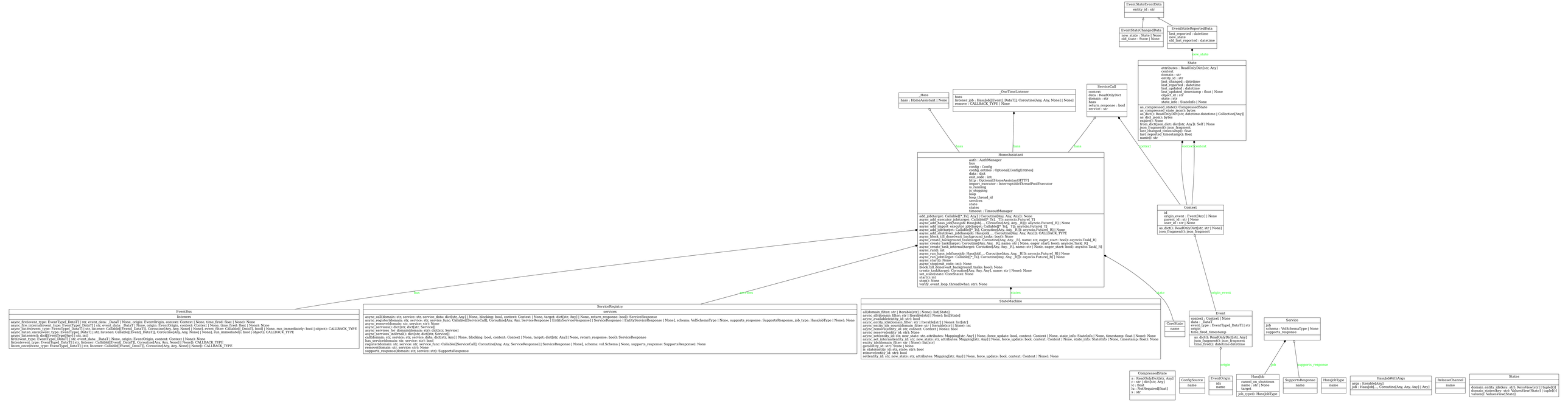
EventStateEventData
entity_id : str
EventStateChangedData
new_state : State | None
old_state : State | None
EventStateReportedData
last_reported : datetime
new_state
old_last_reported : datetime
new_state
State
attributes : ReadOnlyDict[str, Any]
context
domain : str
entity_id : str
last_changed : datetime
last_reported : datetime
last_updated : datetime
last_updated_timestamp : float | None
object_id : str
state : str
state_info : StateInfo | None
as_compressed_state(): CompressedState
as_compressed_state_json(): bytes
as_dict(): ReadOnlyDict[str, datetime.datetime | Collection[Any]]
as_dict_json(): bytes
expired(*, time): bool
from_dictjson(cls, dict: dict[str, Any]): Self | None
json_fragment(): json_fragment
last_changed_timestamp(): float
last_reported_timestamp(): float
name(): str
Hass
hass : HomeAssistant | None
OneTimeListener
hass
listener_job : HassJob[[Event[_DataT]], Coroutine[Any, Any, None] | None]
remove : CALLBACK_TYPE | None
ServiceCall
context
data : ReadOnlyDict
domain : str
hass
return_response : bool
service : str
hass
hass
hass
context
context
context
HomeAssistant
auth : AuthManager
bus
config : Config
config_entries : OptionalConfigEntries]
data : dict
exit_code : int
http : OptionalHomeAssistantHTTP]
loop_executor : InterruptibleThreadPoolExecutor
is_running
is_stopping
loop
loop_thread_id
services
state
states
timeout : TimeoutManager
add_job(target: Callable[*, Ts], Any) | Coroutine[Any, Any, Any]): None
async_add_executor_job(target: Callable[*, Ts], _T): asyncio.Future[_T]
async_add_hass_job(hassjob: HassJob, _Coroutine[Any, Any, _R]): asyncio.Future[_R] | None
async_add_job(target: Coroutine[Any, Any, _R]): asyncio.Future[_R] | None
async_add_job(target: Callable[*, Ts], Coroutine[Any, Any, _R]): asyncio.Future[_R] | None
async_add_shutdown_job(hassjob: HassJob[[_], _Coroutine[Any, Any, Any]]): CALLBACK_TYPE
async_block_till_done(wait_background_tasks: bool): None
async_create_background_task(target: Coroutine[Any, Any, _R], name: str, eager_start: bool): asyncio.Task[_R]
async_create_task(target: Coroutine[Any, Any, _R], name: str | None, eager_start: bool): asyncio.Task[_R]
async_create_task_internal(target: Coroutine[Any, Any, _R], name: str | None, eager_start: bool): asyncio.Task[_R]
async_run(): int
async_run_hass_job(hassjob: HassJob[[_], _Coroutine[Any, Any, _R]]): asyncio.Future[_R] | None
async_run_job(target: Callable[*, Ts], Coroutine[Any, Any, _R]]): asyncio.Future[_R] | None
async_start(): None
async_stop(exit_code: int): None
block_till_done(wait_background_tasks: bool): None
create_task(target: Coroutine[Any, Any, Any], name: str | None): None
set_state(state: CoreState): None
stop(): None
stop(): None
verify_event_loop_thread(what: str): None
Context
id
origin_event : Event[Any] | None
parent_id : str | None
user_id : str | None
as_dict(): ReadOnlyDict[str, str | None]
json_fragment(): json_fragment
bus
services
states
state
origin_event
EventBus
listeners
async_fire(event_type: EventType[_DataT] | str, event_data: _DataT | None, origin: EventOrigin, context: Context | None, time_fired: float | None): None
async_fire_internal(event_type: EventType[_DataT] | str, event_data: _DataT | None, origin: EventOrigin, context: Context | None, time_fired: float | None): None
async_listen_once(event_type: EventType[_DataT] | str, listener: Callable[[Event[_DataT]], Coroutine[Any, Any, None] | None], event_filter: Callable[[_DataT], bool] | None, run_immediately: bool | object): CALLBACK_TYPE
async_listen_once(event_type: EventType[_DataT] | str, listener: Callable[[Event[_DataT]], Coroutine[Any, Any, None] | None], run_immediately: bool | object): CALLBACK_TYPE
async_listeners(): dict[EventType[Any] | str, int]
fire(event_type: EventType[_DataT] | str, event_data: _DataT | None, origin: EventOrigin, context: Context | None): None
listen(event_type: EventType[_DataT] | str, listener: Callable[[Event[_DataT]], Coroutine[Any, Any, None] | None]): CALLBACK_TYPE
listen_once(event_type: EventType[_DataT] | str, listener: Callable[[Event[_DataT]], Coroutine[Any, Any, None] | None]): CALLBACK_TYPE
ServiceRegistry
services
async_call(domain: str, service: str, service_data: dict[str, Any] | None, blocking: bool, context: Context | None, target: dict[str, Any] | None, return_response: bool): ServiceResponse
async_register(domain: str, service: str, service_func: Callable[[ServiceCall], Coroutine[Any, Any, ServiceResponse | EntityServiceResponse] | ServiceResponse | EntityServiceResponse] | None], schema: VolSchemaType | None, supports_response: SupportsResponse, job_type: HassJobType | None): None
async_remove(domain: str, service: str): None
async_services(): dict[str, dict[str, Service]]
async_services_for_domain(domain: str): dict[str, Service]
async_services_internal(): dict[str, dict[str, Service]]
async_validate_service_call(domain: str, service: str, data: dict[str, Any] | None, blocking: bool, context: Context | None, target: dict[str, Any] | None, return_response: bool): ServiceResponse
has_service(domain: str, service: str): bool
register(domain: str, service: str, service_func: Callable[[ServiceCall], Coroutine[Any, Any, ServiceResponse] | ServiceResponse | None], schema: vol.Schema | None, supports_response: SupportsResponse): None
remove(domain: str, service: str): None
supports_response(domain: str, service: str): SupportsResponse
StateMachine
states
all(domain_filter: str | Iterable[str] | None): list[State]
async_all(domain_filter: str | Iterable[str] | None): list[State]
async_available(entity_id: str): bool
async_entity_ids(domain_filter: str | Iterable[str] | None): list[str]
async_entity_ids_count(domain_filter: str | Iterable[str] | None): int
async_remove(entity_id, context): bool | None
async_reserve(entity_id: str): None
async_set(entity_id: str, new_state: str, attributes: Mapping[str, Any] | None, force_update: bool, context: Context | None, state_info: StateInfo | None, timestamp: float | None): None
async_set_internal(entity_id: str, new_state: str, attributes: Mapping[str, Any] | None, force_update: bool, context: Context | None, state_info: StateInfo | None, timestamp: float): None
entity_ids(domain_filter: str | None): list[str]
get(entity_id: str): State | None
is_state(entity_id: str, state: str): bool
remove(entity_id: str): bool
set(entity_id: str, new_state: str, attributes: Mapping[str, Any] | None, force_update: bool, context: Context | None): None
CoreState
name
Event
context : Context | None
data : _DataT
event_type : EventType[_DataT] | str
origin
time_fired_timestamp
as_dict(): ReadOnlyDict[str, Any]
json_fragment(): json_fragment
time_fired(): datetime.datetime
Service
job
schema : VolSchemaType | None
supports_response
CompressedState
a : ReadOnlyDict[str, Any]
c : str | dictstr, Any]
lc : float
s : str | NotRequired[float]
u : str
ConfigSource
name
EventOrigin
idx
name
HassJob
cancel_on_shutdown
name : str | None
target
job_type(): HassJobType
SupportsResponse
name
HassJobType
name
HassJobWithArgs
args : Iterable[Any]
job : HassJob[[_], _Coroutine[Any, Any, Any]] | Any]
ReleaseChannel
name
States
domain_entity_idx(key: str): KeysView[str] | tuple[()]
domain_states(key: str): ValuesView[State] | tuple[()]
values(): ValuesView[State]